8051 CROSS ASSEMBLER

USER'S MANUAL

MetaLink Corporation
Chandler, Arizona

MetaLink Corporation
Chandler, Arizona USA
www.metaice.com

NOTICE

MetaLink Corp. reserves the right to make improvements in the software product described in this manual as well as the manual itself at any time and without notice.

DISCLAIMER OF ALL WARRANTIES AND LIABILITY

METALINK CORP. MAKES NO WARRANTIES, EITHER EXPRESSED OR IMPLIED, WITH RESPECT TO THIS MANUAL OR WITH RESPECT TO THE SOFTWARE DESCRIBED IN THIS MANUAL, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. METALINK CORP. SOFTWARE IS SOLD, LICENSED OR DISTRIBUTED "AS IS". IN NO EVENT SHALL METALINK CORP. BE LIABLE FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE SOFTWARE.

T A B L E   O F   C O N T E N T S

L I S T O F T A B L E S

CHAPTER 1

1 8051 OVERVIEW

1.1 Introduction

For new users of MetaLink's ASM51 Cross Assembler, please take a moment to fill out and return postage-prepaid User Registration Card. This card will be found with the system diskette in the vinyl jacket at the front of this manual. MetaLink will use this information to send out, free of charge, any software updates occurring during the warranty period. Respondents will also be notified of any new products or product enhancements.

The 8051 series of microcontrollers are highly integrated single chip microcomputers with an 8-bit CPU, memory, interrupt controller, timers, serial I/O and digital I/O on a single piece of silicon. The current members of the 8051 family of components include:

80C152JA/JB/JC/JD, 83C152JA/JC, 80C157
80C154, 83C154, 85C154
8044, 8344, 8744
80C451, 83C451, 87C451
80C452, 83C452, 87C452
8051, 8031, 8751, 80C51, 80C31, 87C51
80512, 80532
80515, 80535, 80C535, 80C515
80C517, 80C537
80C51FA, 83C51FA, 87C51FA, 83C51FB, 87C51FB, 83C51FC, 87C51FC
8052, 8032, 8752
80C321, 80C521, 87C521, 80C541, 87C541
8053, 9761, 8753
80C552, 83C552, 87C552
80C652, 83C652, 87C652
83C654, 87C654
83C751, 87C751
83C752, 87C752
80C851, 83C851

All members of the 8051 series of microcontrollers share a common architecture. They all have the same instruction set, addressing modes, addressing range and memory spaces. The primary differences between different 8051 based products are the amount of memory on chip, the amount and types of I/O and peripheral functions, and the component's technology (see Table 1-1).

In the brief summary of the 8051 architecture that follows, the term 8051 is used to mean collectively all available members of the 8051 family. Please refer to reference (1) for a complete description of the 8051 architecture and the specifications for all the currently available 8051 based products.

1.2 8051 Architecture

The 8051 is an 8-bit machine. Its memory is organized in bytes and practically all its instruction deal with byte quantities. It uses an Accumulator as the primary register for instruction results. Other operands can be accessed using one of the four different addressing modes available: register implicit, direct, indirect or immediate. Operands reside in one of the five memory spaces of the 8051.

The five memory spaces of the 8051 are: Program Memory, External Data Memory, Internal Data Memory, Special Function Registers and Bit Memory.

The Program Memory space contains all the instructions, immediate data and constant tables and strings. It is principally addressed by the 16-bit Program Counter (PC), but it can also be accessed by a few instructions using the 16-bit Data Pointer (DPTR). The maximum size of the Program Memory space is 64K bytes. Several 8051 family members integrate on-chip some amount of either masked programmed ROM or EPROM as part of this memory space (refer to Table 1-1).

The External Data Memory space contains all the variables, buffers and data structures that can't fit on-chip. It is principally addressed by the 16-bit Data Pointer (DPTR), although the first two general purpose register (R0,R1) of the currently selected register bank can access a 256-byte bank of External Data Memory. The maximum size of the External Data Memory space is 64Kbytes. External data memory can only be accessed using the indirect addressing mode with the DPTR, R0 or R1.

The Internal Data Memory space is functionally the most important data memory space. In it resides up to four banks of general purpose registers, the program stack, 128 bits of the 256-bit memory, and all the variables and data structures that are operated on directly by the program. The maximum size of the Internal Data Memory space is 256-bytes. However, different 8051 family members integrate different amounts of this memory space on chip (see Amnt of RAM in Table 1-1). The register implicit, indirect and direct addressing modes can be used in different parts of the Internal Data Memory space.

The Special Function Register space contains all the on-chip peripheral I/O registers as well as particular registers that need program access. These registers include the Stack Pointer, the PSW and the Accumulator. The maximum number of Special Function Registers (SFRs) is 128, though the actual number on a particular 8051 family member depends on the number and type of peripheral functions integrated on-chip (see Table 1-1). The SFRs all have addresses greater than 127 and overlap the address space of the upper 128 bytes of the Internal Data Memory space. The two memory spaces are differentiated by addressing mode. The SFRs can only be accessed using the Direct addressing mode while the upper 128 bytes of the Internal Data Memory (if integrated on-chip) can only be accessed using the Indirect addressing mode.

The Bit Memory space is used for storing bit variables and flags. There are specific instructions in the 8051 that operate only in the Bit Memory space. The maximum size of the Bit Memory space is 256-bits. 128 of the bits overlap with 16-bytes of the Internal Data Memory space and 128 of the bits overlap with 16 Special Function Registers. Bits can only be accessed using the bit instructions and the Direct addressing mode.

The 8051 has a fairly complete set of arithmetic and logical instructions. It includes an 8X8 multiply and an 8/8 divide. The 8051 is particularly good at processing bits (sometimes called Boolean Processing). Using the Carry Flag in the PSW as a single bit accumulator, the 8051 can move and do logical operations between the Bit Memory space and the Carry Flag. Bits in the Bit Memory space can also be used as general purpose flags for the test bit and jump instructions.

Except for the MOVE instruction, the 8051 instructions can only operate on either the Internal Data Memory space or the Special Function Registers. The MOVE instruction operates in all memory spaces, including the External Memory space and Program Memory space.

Program control instructions include the usual unconditional calls and jumps as well as conditional relative jumps based on the Carry Flag, the Accumulator's zero

state, and the state of any bit in the Bit Memory space. Also available is a Compare and Jump if Not Equal instruction and a Decrement Counter and Jump if Not Zero loop instruction. See Chapter 4 for a description of the complete 8051 instruction set.

1.3 Summary of the 8051 Family of Components

Table 1-1: 8051 Family of Components

Component	Technology	Amnt of ROM	Type of ROM	Amnt of RAM	No. of SFRs	Serial I/O Type
8031	HMOS	0	--	128 bytes	21	Start/Stop Async
8051	HMOS	4Kbytes	Masked	128 bytes	21	Start/Stop Async
8751	HMOS	4Kbytes	EPROM	128 bytes	21	Start/Stop Async
8053	HMOS	8Kbytes	Masked	128 bytes	21	Start/Stop Async
9761	HMOS	8Kbytes	EPROM	128 bytes	21	Start/Stop Async
8751	HMOS	8Kbytes	EPROM	128 bytes	21	Start/Stop Async
80C31	CMOS	0	--	128 bytes	21	Start/Stop Async
80C51	CMOS	4Kbytes	Masked	128 bytes	21	Start/Stop Async
87C51	CMOS	4Kbytes	EPROM	128 bytes	21	Start/Stop Async
8032	HMOS	0	--	256 bytes	26	Start/Stop Async
8052	HMOS	8Kbytes	Masked	256 bytes	26	Start/Stop Async
8752	HMOS	8Kbytes	EPROM	256 bytes	26	Start/Stop Async
80C32	CMOS	0	--	256 bytes	26	Start/Stop Async
80C52	CMOS	8Kbytes	Masked	256 bytes	26	Start/Stop Async
87C52	CMOS	8Kbytes	EPROM	256 bytes	26	Start/Stop Async
8044	HMOS	4Kbytes	Masked	192 bytes	34	HDLC/SDLC
8344	HMOS	0	--	192 bytes	34	HDLC/SDLC
8744	HMOS	4Kbytes	EPROM	192 bytes	34	HDLC/SDLC
80535	HMOS	0	--	256 bytes	41	Start/Stop Async
80515	HMOS	8Kbytes	Masked	256 bytes	41	Start/Stop Async
80C535	CHMOS	0	--	256 bytes	41	Start/Stop Async
80C515	CHMOS	8Kbytes	Masked	256 bytes	41	Start/Stop Async
80532	HMOS	0	--	128 bytes	28	Start/Stop Async
80512	HMOS	4Kbytes	Masked	128 bytes	28	Start/Stop Async
80C152	CHMOS	0	--	256 bytes	56	CSMA/CD
83C152	CHMOS	8Kbytes	Masked	256 bytes	56	CSMA/CD
80C154	CMOS	0	--	256 bytes	27	Start/Stop Async
83C154	CMOS	16Kbytes	Masked	256 bytes	27	Start/Stop Async
85C154	CMOS	16Kbytes	EPROM	256 bytes	27	Start/Stop Async
80C51FA	CHMOS	0	--	256 bytes	47	Start/Stop Async
83C51FA	CHMOS	8Kbytes	Masked	256 bytes	47	Start/Stop Async
87C51FA	CHMOS	8Kbytes	EPROM	256 bytes	47	Start/Stop Async
83C51FB	CHMOS	16Kbytes	Masked	256 bytes	47	Start/Stop Async
87C51FB	CHMOS	16Kbytes	EPROM	256 bytes	47	Start/Stop Async
83C51FB	CHMOS	32Kbytes	Masked	256 bytes	47	Start/Stop Async
87C51FB	CHMOS	32Kbytes	EPROM	256 bytes	47	Start/Stop Async
80C537	CHMOS	0	--	256 bytes	41	Start/Stop Async
80C517	CHMOS	8Kbytes	Masked	256 bytes	82	Start/Stop Async
80C451	CMOS	0	--	128 bytes	24	Parallel I/F
83C451	CMOS	4Kbytes	Masked	128 bytes	24	Parallel I/F
87C451	CMOS	4Kbytes	EPROM	128 bytes	24	Parallel I/F
80C452	CHMOS	0	--	256 bytes	55	U.P.I.
83C452	CHMOS	8Kbytes	--	256 bytes	55	U.P.I.
87C452	CHMOS	8Kbytes	--	256 bytes	55	U.P.I.
80C552	CMOS	0	--	256 bytes	54	Start/Stop Async
83C552	CMOS	8Kbytes	Masked	256 bytes	54	Start/Stop Async
87C552	CMOS	8Kbytes	EPROM	256 bytes	54	Start/Stop Async
80C652	CMOS	0	--	256 bytes	24	Start/Stop Async
83C652	CMOS	8Kbytes	Masked	256 bytes	24	Start/Stop Async
87C652	CMOS	8Kbytes	EPROM	256 bytes	24	Start/Stop Async
83C654	CMOS	16Kbytes	Masked	256 bytes	24	Start/Stop Async

Component	Technology	Amnt of ROM	Type of ROM	Amnt of RAM	No. of SFRs	Serial I/O Type
87C654	CMOS	16Kbytes	EPROM	256 bytes	24	Start/Stop Async
83C752	CMOS	2Kbytes	Masked	64 bytes	25	I2C
87C752	CMOS	2Kbytes	EPROM	64 bytes	25	I2C
83C751	CMOS	2Kbytes	Masked	64 bytes	20	I2C
87C751	CMOS	2Kbytes	EPROM	64 bytes	20	I2C
80C521	CMOS	0	--	256 bytes	26	Start/Stop Async
80C321	CMOS	8Kbytes	Masked	256 bytes	26	Start/Stop Async
87C521	CMOS	8Kbytes	EPROM	256 bytes	26	Start/Stop Async
80C541	CMOS	16Kbytes	Masked	256 bytes	26	Start/Stop Async
87C541	CMOS	16Kbytes	EPROM	256 bytes	26	Start/Stop Async
80C851	CMOS	0	--	128 bytes	21	Start/Stop Async
83C851	CMOS	4Kbytes	Masked	128 bytes	21	Start/Stop Async

1.4 References

1. Intel Corp., 8-Bit Embedded Controllers, 1990.
2. Siemens Corp., Microcontroller Component 80515, 1985.
3. AMD Corp., Eight-Bit 80C51 Embedded Processors, 1990.
4. Signetics Corp., Microcontroller Users' Guide, 1989.

CHAPTER 2

2 8051 CROSS ASSEMBLER OVERVIEW

2.1 Introduction

The 8051 Cross Assembler takes an assembly language source file created with a text editor and translates it into a machine language object file. This translation process is done in two passes over the source file. During the first pass, the Cross Assembler builds a symbol table from the symbols and labels used in the source file. It's during the second pass that the Cross Assembler actually translates the source file into the machine language object file. It is also during the second pass that the listing is generated.

The following is a discussion of the syntax required by the Cross Assembler to generate error free assemblies.

2.2 Symbols

Symbols are alphanumeric representations of numeric constants, addresses, macros, etc. The legal character set for symbols is the set of letters, both upper and lower case (A..Z,a..z), the set of decimal numbers (0..9) and the special characters, question mark (?) and underscore (_). To ensure that the Cross Assembler can distinguish between a symbol and a number, all symbols must start with either a letter or special character (? or _). The following are examples of legal symbols:

```
PI
Serial_Port_Buffer
LOC_4096
?_?_?
```

In using a symbol, the Cross Assembler converts all letters to upper case. As a result, the Cross Assembler makes no distinction between upper and lower case letters. For example, the following two symbols would be seen as the same symbol by the Cross Assembler:

```
Serial_Port_Buffer
SERIAL_PORT_BUFFER
```

Symbols can be defined only once. Symbols can be up to 255 characters in length, though only the first 32 are significant. Therefore, for symbols to be unique, they must have a unique character pattern within the first 32 characters. In the following example, the first two symbols would be seen by the Cross Assembler as duplicate symbols, while the third and fourth symbols are unique.

```
BEGINNING_ADDRESS_OF_CONSTANT_TABLE_1
BEGINNING_ADDRESS_OF_CONSTANT_TABLE_2

CONSTANT_TABLE_1_BEGINNING_ADDRESS
CONSTANT_TABLE_2_BEGINNING_ADDRESS
```

There are certain symbols that are reserved and can't be defined by the user. These reserved symbols are listed in Appendix C and include the assembler directives, the 8051 instruction mnemonics, implicit operand symbols, and the following assembly time operators that have alphanumeric symbols: EQ, NE, GT, GE, LT, LE, HIGH, LOW, MOD, SHR, SHL, NOT, AND, OR and XOR.

The reserved implicit operands include the symbols A, AB, C, DPTR, PC, R0, R1, R2, R3, R4, R5, R6, R7, AR0, AR1, AR2, AR3, AR4, AR5, AR6 and AR7. These symbols are used primarily as instruction operands. Except for AB, C, DPTR or PC, these symbols can also be used to define other symbols (see EQU directive in Chapter 5).

The following are examples of illegal symbols with an explanation of why they are illegal:

```
      1ST_VARIABLE      (Symbols can not start with a number.)
      ALPHA#            (Illegal character "#" in symbol.)
      MOV               (8051 instruction mnemonic)
      LOW               (Assembly operator)
      DATA              (Assembly directive)
```

2.3 Labels

Labels are special cases of symbols. Labels are used only before statements that have physical addresses associated with them. Examples of such statements are assembly language instructions, data storage directives (DB and DW), and data reservation directives (DS and DBIT). Labels must follow all the rules of symbol creation with the additional requirement that they be followed by a colon. The following are legal examples of label uses:

```
TABLE_OF_CONTROL_CONSTANTS:
      DB    0,1,2,3,4,5     (Data storage)
MESSAGE:    DB    'HELP'    (Data storage)
VARIABLES:  DS    10        (Data reservation)
BIT_VARIABLES:    DBIT  16  (Data reservation)
START:      MOV   A,#23     (Assembly language instruction)
```

2.4 Assembler Controls

Assembler controls are used to control where the Cross Assembler gets its input source file, where it puts the object file, and how it formats the listing file. Table 2-1 summarizes the assembler controls available. Refer to Chapter 6 for a detailed explanation of the controls.

Table 2-1: Summary of Cross Assembler Controls

Control	Description	Control	Description
$DATE(date)	Places date in page header	$EJECT	Places a form feed in listing
$INCLUDE(file)	Inserts file in source program	$LIST	Allows listing to be output
$NOLIST	Stops outputting the listing	$MOD51	Uses 8051 predefined symbols
$MOD52	Uses 8052 predefined symbols	$MOD44	Uses 8044 predefined symbols
$NOMOD	No predefined symbols used	$OBJECT(file)	Places object output in file
$NOOBJECT	No object file is generated	$PAGING	Break output listing into pages
$NOPAGING	Print listing w/o page breaks	$PAGELENGTH(n)	No. of lines on a listing page
$PAGEWIDTH(n)	No. of columns on a listing page	$PRINT(file)	Places listing output in file
$NOPRINT	Listing will not be output	$SYMBOLS	Append symbol table to listing
$NOSYMBOLS	Symbol table will not be output	$TITLE(string)	Places string in page header

As can be seen in Table 2-1, all assembler controls are prefaced with a dollar sign ($). No spaces or tabs are allowed between the dollar sign and the body of the control. Also, only one control per line is permitted. However, comments can be on the same line as a control. The following are examples of assembler controls:

```
$TITLE(8051 Program Ver. 1.0)
$LIST
$PAGEWIDTH(132)
```

2.5 Assembler Directives

Assembler directives are used to define symbols, reserve memory space, store values in program memory and switch between different memory spaces. There are also directives that set the location counter for the active segment and identify the end of the source file. Table 2-2 summarizes the assembler directives available. These directives are fully explained in Chapter 5.

Table 2-2: Summary of Cross Assembler Directives

Directive	Description	Directive	Description
EQU	Define symbol	DATA	Define internal memory symbol
IDATA	Define indirectly addressed internal memory symbol	XDATA	Define external memory symbol
		BIT	Define internal bit memory symbol
CODE	Define program memory symbol	DS	Reserve bytes of data memory
DBIT	Reserve bits of bit memory	DB	Store byte values in program memory
DW	Store word values in program memory	ORG	Set segment location counter
END	End of assembly language source file	CSEG	Select program memory space
		XSEG	Select external memory data space
DSEG	Select internal memory data space	BSEG	Select bit addressable memory space
ISEG	Select indirectly addressed internal memory space	USING	Select register bank
		ELSE	Alternative conditional assembly block
IF	Begin conditional assembly block		
ENDIF	End conditional assembly block		

Only one directive per line is allowed, however comments may be included. The following are examples of assembler directives:

```
TEN       EQU       10
RESET     CODE      0
ORG       4096
```

2.6 8051 Instruction Mnemonics

The standard 8051 Assembly Language Instruction mnemonics plus the generic CALL and JMP instructions are recognized by the Cross Assembler and are summarized in Table 2-3. See Chapter 4 for the operation of the individual instructions.

Table 2-3: 8051 Instructions and Mnemonics

Mnemonic	Description	Mnemonic	Description	Mnemonic	Description
ACALL	Absolute call	ADD	Add	ADDC	Add with carry
AJMP	Absolute jump	ANL	Logical and	CJNE	Compare, jump if not equal
CLR	Clear	CPL	Complement	DA	Decimal adjust
DEC	Decrement	DIV	Divide	DJNZ	Decrement, jump if not zero
INC	Increment	JB	Jump if bit set	JBC	Jump & clear bit if bit set
JC	Jump if carry set	JMP	Jump	JNB	Jump if bit not set
JNC	Jump if carry not set	JNZ	Jump if accum. not zero	JZ	Jump if accumulator zero
LCALL	Long call	LJMP	Long jump	MOV	Move
MOVC	Move code	MOVX	Move external	MUL	Multiply
NOP	No operation	ORL	Inclusive or	POP	Pop stack
PUSH	Push stack	RET	Return	RETI	Return from interrupt
RL	Rotate left	RLC	Rotate left through carry	RR	Rotate right
RRC	Rotate right through carry	SETB	Set bit	SJMP	Short jump
SUBB	Subtract with borrow	SWAP	Swap nibbles	XCH	Exchange bytes
XCHD	Exchange digits	XRL	Exclusive or	CALL	Generic call

When the Cross Assembler sees a generic CALL or JMP instruction, it will try to translate the instruction into its most byte efficient form. The Cross Assembler will translate a CALL into one of two instructions (ACALL or LCALL) and it will translate a generic JMP into one of three instructions (SJMP, AJMP or LJMP). The choice of instructions is based on which one is most byte efficient. The generic CALL or JMP instructions saves the programmer the trouble of determining which form is best.

However, generic CALLs and JMPs do have their limitations. While the byte efficiency algorithm works well for previously defined locations, when the target location of the CALL or JMP is a forward location (a location later on in the program), the assembler has no way of determining the best form of the instruction. In this case the Cross Assembler simply puts in the long version (LCALL or LJMP) of the instruction, which may not be the most byte efficient. NOTE that the generic CALLs and JMPs must not be used for the 751/752 device as LCALL and LJMP are not legal instructions for those devices. Instead use ACALL and AJMP explicitly.

For instructions that have operands, the operands must be separated from the mnemonic by at least one space or tab. For instructions that have multiple operands, each operand must be separated from the others by a comma.

Two addressing modes require the operands to be preceded by special symbols to designate the addressing mode. The AT sign (@) is used to designate the indirect addressing mode. It is used primarily with Register 0 and Register 1 (R0, R1), but is can also be used with the DPTR in the MOVX and the Accumulator in MOVC and JMP @A+DPTR instructions. The POUND sign (#) is used to designate an immediate operand. It can be used to preface either a number or a symbol representing a number.

A third symbol used with the operands actually specifies an operation. The SLASH (/) is used to specify that the contents of a particular bit address is to be complemented before the instruction operation. This is used with the ANL and ORL bit instructions.

Only one assembly language instruction is allowed per line. Comments are allowed on the same line as an instruction, but only after all operands have been specified. The following are examples of instruction statements:

```
START:    LJMP     INIT
          MOV      @R0,Serial_Port_Buffer
          CJNE     R0 , #TEN, INC_TEN
          ANL      C,/START_FLAG
          CALL     GET_BYTE
          RET
```

2.7 Bit Addressing

The period (.) has special meaning to the Cross Assembler when used in a symbol. It is used to explicitly specify a bit in a bit-addressable symbol. For example, it you wanted to specify the most significant bit in the Accumulator, you could write ACC.7, where ACC was previously defined as the Accumulator address. The same bit can also be selected using the physical address of the byte it's in. For example, the Accumulator's physical address is 224. The most significant bit of the Accumulator can be selected by specifying 224.7. If the symbol ON was defined to be equal to the value 7, you could also specify the same bit by either ACC.ON or 224.ON.

2.8 ASCII Literals

Printable characters from the ASCII character set can be used directly as an immediate operand, or they can used to define symbols or store ASCII bytes in Program Memory. Such use of the ASCII character set is called ASCII literals. ASCII literals are identified by the apostrophe (') delimiter. The apostrophe itself can be used as an ASCII literal. In this case, use two apostrophes in a row. Below are examples of using ASCII literals.

```
          MOV  A, #'m'   ;Load A with 06DH (ASCII m)
QUOTE     EQU  ''''      ;QUOTE defined as 27H (ASCII single quote)
          DB   '8051'    ;Store in Program Memory
```

2.9 Comments

Comments are user defined character strings that are not processed by the Cross Assembler. A comment begins with a semicolon (;) and ends at the carriage return/line feed pair that terminates the line. A comment can appear anywhere in a line, but it has to be the last field. The following are examples of comment lines:

```
; Begin initialization routine here
$TITLE(8051 Program Vers. 1.0)          ;Place version number here
TEN       EQU      10                   ;Constant
          ; Comment can begin anywhere in a line
          MOV  A, Serial_Port_Buffer          ; Get character
```

2.10 The Location Counter

The Cross Assembler keeps a location counter for each of the five segments (code, internal data, external data, indirect internal data and bit data). Each location counter is initialized to zero and can be modified using Assembler Directives described in Chapter 5.

The dollar sign ($) can be used to specify the current value of the location counter of the active segment. The following are examples of how this can be used:

```
          JNB  FLAG,$        ;Jump on self until flag is reset
CPYRGHT:  DB   'Copyright, 1983'
CPYRGHT_LENGTH
          EQU  $-CPYRGHT-1   ;Calculate length of copyright message
```

2.11 Syntax Summary

Since the Cross Assembler essentially translates the source file on a line by line basis, certain rules must be followed to ensure the translation process is done correctly. First of all, since the Cross Assembler's line buffer is 256 characters deep, there must always be a carriage return/line feed pair within the first 256 columns of the line.

A legal source file line must begin with either a control, a symbol, a label, an instruction mnemonic, a directive, a comment or it can be null (just the carriage return/line feed pair). Any other beginning to a line will be flagged as an error.

While a legal source file line must begin with one of the above items, the item doesn't have to begin in the first column of the line. It only must be the first field of the line. Any number (including zero) of spaces or tabs, up to the maximum line size, may precede it.

Comments can be placed anywhere, but they must be the last field in any line.

2.12 Numbers and Operators

The Cross Assembler accepts numbers in any one of four radices: binary, octal, decimal and hexadecimal. To specify a number in a specific radix, the number must use the correct digits for the particular radix and immediately following the number with its radix designator. Decimal is the default radix and the use of its designator is optional. An hexadecimal number that would begin with a letter digit must be preceded by a 0 (zero) to distinguish it from a symbol. The internal representation of numbers is 16-bits, which limits the maximum number possible. Table 2-4 summarizes the radices available.

Table 2-4: Cross Assembler Radices

MAXIMUM LEGAL RADIX	DESIGNATOR	LEGAL DIGITS	NUMBER
Binary	B	0,1	1111111111111111 B
Octal	O,Q	0,1,2,3,4,5,6,7	177777O 177777Q
Decimal	D,(default)	0,1,2,3,4,5,6,7,8,9	65535D 65535
Hexadecimal	H	0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F	0FFFFH

No spaces or tabs are allowed between the number and the radix designator. The letter digits and radix designators can be in upper or lower case. The following examples list the decimal number 2957 in each of the available radices:

```
101110001101B     (Binary)
5615o or 5615Q    (Octal)
2957 or 2957D     (Decimal)
0B8DH, 0b8dh      (Hexadecimal)
```

When using radices with explicit bit symbols, the radix designator follows the byte portion of the address as shown in the following examples:

```
0E0H.7            Bit seven of hexadecimal address 0E0
200Q.ON           Bit ON of octal address 200
```

The Cross Assembler also allows assembly time evaluation of arithmetic expressions up to thirty-two levels of embedded parentheses. All calculations use integer numbers and are done in sixteen bit precision.

Table 2-5: Assembly Time Operations

OPERATOR SYMBOL	OPERATION
+	Addition Unary positive
-	Subtraction Unary negation (2's complement)
*	Multiplication

OPERATOR SYMBOL	OPERATION
/	Integer division (no remainder)
MOD	Modulus (remainder of integer division)
SHR	Shift right
SHL	Shift left
NOT	Logical negation (1's complement)
AND	Logical and
OR	Inclusive or
XOR	Exclusive or
LOW	Low order 8-bits
HIGH	High order 8-bits
EQ, =	Relational equal
NE, <>	Relational not equal
GT, >	Relational greater than
GE, >=	Relational greater than or equal
LT, <	Relational less than
LE, <=	Relational less than or equal
( )	Parenthetical statement

The relational operators test the specified values and return either a True or False. False is represented by a zero value, True is represented by a non zero value (the True condition actually returns a 16-bit value with every bit set; i.e., 0FFFFH). The relational operators are used primarily with the Conditional Assembly capability of the Cross Assembler.

Table 2-5 lists the operations available while Table 2-6 lists the operations precedence in descending order. Operations with higher precedence are done first. Operations with equal precedence are evaluated from left to right.

Table 2-6: Operators Precedence

OPERATION	PRECEDENCE
(,)	HIGHEST
HIGH,LOW	
*,/,MOD,SHR,SHL	
+,-	
EQ,LT,GT,LE,GE,NE,=,<,>,<=,>=,<>	
NOT	
AND	
OR,XOR	LOWEST

The following are examples of all the available operations and their result:

```
HIGH(0AADDH)       will return a result of 0AAH
LOW(0AADDH)        will return a result of 0DDH
7*4                will return a result of 28
7/4                will return a result of 1
7 MOD 4            will return a result of 3
1000B SHR 2        will return a result of 0010B
1010B SHL 2        will return a result of 101000B
10+5               will return a result of 15
```

```
+72               will return a result of 72
25-17             will return a result of 8
-1                will return a result of 1111111111111111B
NOT 1             will return a result of 1111111111111110B
7 EQ 4, 7 = 4     will return a result of 0
7 LT 4, 7 < 4     will return a result of 0
7 GT 4, 7 > 4     will return a result of 0FFFFH
7 LE 4, 7 <= 4    will return a result of 0
7 GE 4, 7 >= 4    will return a result of 0FFFFH
7 NE 4, 7 <> 4    will return a result of 0FFFFH
1101B AND 0101B   will return a result of 0101B
1101B OR 0101B    will return a result of 1101B
1101B XOR 0101B   will return a result of 1000B
```

2.13 Source File Listing

The source file listing displays the results of the Cross Assembler translation. Every line of the listing includes a copy of the original source line as well as a line number and the Cross Assembler translation.

For example, in translating the following line taken from the middle of a source file:

```
TRANS:    MOV  R7,#32         ;Set up pointer
```

the listing will print:

```
002F 7920         152   TRANS:  MOV   R1,#32        ;Set up pointer
```

The '002F' is the current value of the location counter in hexadecimal. The '7920' is the translated instruction, also in hexadecimal. The '152' is the decimal line number of the current assembly. After the line number is a copy of the source file line that was translated.

Another example of a line in the listing file is as follows:

```
015B 13      =1 267 +2          RRC   A
```

Here we see two additional fields. The '=1' before the line number gives the current nesting of include files. The '+2' after the line number gives the current macro nesting. This line essentially says that this line comes from a second level nesting of a macro that is part of an include file.

Another line format that is used in the listing is that of symbol definition. In this case the location counter value and translated instruction fields described above are replaced with the definition of the symbol. The following are examples of this:

```
00FF              67   MAX_NUM      EQU       255
REG               68   COUNTER      EQU       R7
```

The '00FF' is the hexadecimal value of the symbol MAX_NUM. Again, '67'is the decimal line number of the source file and the remainder of the first line is a copy of the source file. In the second line above, the 'REG' shows that the symbol COUNTER was defined to be a general purpose register.

Optionally, a listing can have a page header that includes the name of the file being assembled, title of program, date and page number. The header and its fields are controlled by specific Assembler Controls (see Chapter 6).

The default case is for a listing to be output as a file on the default drive with the same name as the entered source file and an extension of .LST. For example, if the source file name was PROGRAM.ASM, the listing file would be called PROGRAM.LST. Or if the source file was called MODULE1, the listing file would be stored as MODULE1.LST. The default can be changed using the $NOPRINT and $PRINT() Assembler Controls (see Chapter 6).

2.14 Object File

The 8051 Cross Assembler also creates a machine language object file. The format of the object file is standard Intel Hexadecimal. This Hexadecimal file can be used to either program EPROMs using standard PROM Programmers for prototyping, or used to pattern masked ROMs for production.

The default case is for the object file to be output on the default drive with the same name as the first source file and an extension of .HEX. For example, if the source file name was PROGRAM.ASM, the object file would be called PROGRAM.HEX. Or if the source file was called MODULE1, the object file would be stored as MODULE1.HEX. The default can be changed using the $NOOBJECT and $OBJECT() Assembler Controls (see Chapter 6).

CHAPTER 3

3 RUNNING THE 8051 CROSS ASSEMBLER ON PC-DOS/MS-DOS SYSTEMS

3.1 Cross Assembler Files

The floppy disk you receive with this manual is an 8 sector, single-sided, double density disk. This distribution disk will contain the following files:

ASM51.EXE	The Cross Assembler program itself
MOD152	Source file for the $MOD152 control
MOD154	Source file for the $MOD154 control
MOD252	Source file for the $MOD252 control
MOD44	Source file for the $MOD44 control
MOD451	Source file for the $MOD451 control
MOD452	Source file for the $MOD452 control
MOD51	Source file for the $MOD51 control
MOD512	Source file for the $MOD512 control
MOD515	Source file for the $MOD515 control
MOD517	Source file for the $MOD517 control
MOD52	Source file for the $MOD52 control
MOD521	Source file for the $MOD521 control
MOD552	Source file for the $MOD552 control
MOD652	Source file for the $MOD652 control
MOD751	Source file for the $MOD751 control
MOD752	Source file for the $MOD752 control
MOD851	Source file for the $MOD851 control

There will also be one or more files with an extension of .ASM. These are sample programs. Listings of these programs can be found in Appendix A.

DON'T USE THE DISTRIBUTION DISK. MAKE WORKING AND BACKUP COPIES FROM THE DISTRIBUTION DISK AND THEN STORE THE DISTRIBUTION DISK IN A SAFE PLACE.

3.2 Minimum System Requirements

With DOS 2.0 or later - 96K RAM
1 Floppy Disk Drive

3.3 Running the Cross Assembler

Once you've created an 8051 assembly language source text file in accordance with the guidelines in Chapter 2, you are now ready to run the Cross Assembler. Make sure your system is booted and the DOS prompt (A>) appears on the screen. Place the disk with the 8051 Cross Assembler on it in the drive and simply type (in all the following examples, the symbol <CR> is used to show where the ENTER key was hit):

```
ASM51<CR>
```

If the 8051 Cross Assembler disk was placed in a drive other than the default drive, the drive name would have to be typed first. For example, if the A drive is the default drive, and the 8051 Cross Assembler is in the B drive, you would then type:

```
B:ASM51<CR>
```

After loading the program from the disk, the program's name, its version number and general copyright information will be displayed on the screen. The Cross Assembler then asks for the source file name to begin the assembly process.

Source file drive and name [.ASM]:

At this point, if you have only one floppy disk drive and the 8051 Cross Assembler and source files are on separate disks, remove the disk with the 8051 Cross Assembler on it and replace it with your source file disk.

Next, enter the source file name. If no extension is given, the Cross Assembler will assume an extension of .ASM. If no drive is given, the Cross Assembler will assume the default drive. Since in every case where no drive is given, the Cross Assembler assumes the default drive, it is generally a good practice to change the default drive to the drive with your source files.

An alternative method for entering the source file is in the command line. In this case, after typing in ASM51, type in a space and the source file name (again if no extension is given, source file on the command line:

A>ASM51 B:CONTROL.A51<CR>

After the source file name has been accepted, the Cross Assembler will begin the translation process. As it starts the first pass of its two pass process, it will print on the screen:

First pass

At the completion of the first pass, and as it starts its second pass through the source file, the Cross Assembler will display:

Second pass

When second pass is completed, the translation process is done and the Cross Assembler will print the following message:

ASSEMBLY COMPLETE, XX ERRORS FOUND

XX is replaced with the actual number of errors that were found. Disk I/O may continue for a while as the Cross Assembler appends the symbol table to the listing file.

3.4 Example Running the Cross Assembler

The following is an example of an actual run. The Cross Assembler will take the source file SAMPLE.ASM from Drive A (default drive).

Again, the symbol <CR> is used to show where the ENTER key was hit.

```
>ASM51<CR>

                  8 0 5 1   C R O S S   A S S E M B L E R

                              Version  1.2

            (c) Copyright 1984, 1985, 1986, 1987, 1988, 1989, 1990

                            MetaLink Corporation


Source file drive and name [.ASM]: sample<CR>

First pass

Second pass

ASSEMBLY COMPLETE, 0 ERRORS FOUND
```

3.5 DOS Hints and Suggestions

If you are using DOS 2.0 or later, you may want to use the BREAK ON command before you run the Cross Assembler. This will allow you to abort (Ctrl-Break) the Cross Assembler at any time. Otherwise, you will only be able to abort the Cross Assembler after it completes a pass through the source file. If you are assembling a large file, this could cause you a several minute wait before the Cross Assembler aborts.

The reason for this it that the default condition for DOS to recognizes a Ctrl-Break is when the program (in this case the Cross Assembler) does keyboard, screen or printer I/O. Unfortunately, the assembler does this very rarely (once each pass). By using the BREAK ON command, DOS will recognize a Ctrl-Break for all I/O, including disk I/O. Since the Cross Assembler is constantly doing disk I/O, with BREAK ON you can abort almost immediately by hitting the Ctrl-Break keys.

So much for the good news. However, aborting a program can cause some undesirable side-effects. Aborting a program while files are open causes DOS to drop some information about the open files. This results in disk sectors being allocated when they are actually free. Your total available disk storage shrinks. You should make the practice of running CHKDSK with the /F switch periodically to recover these sectors.

The Cross Assembler run under DOS 2.0 or later supports redirection. You can specify the redirection on the command line. Use the following form:

```
ASM51 <infile >outfile
```

"infile" and "outfile" can be any legal file designator. The Cross Assembler will take its input from the "infile" instead of the keyboard and will send its output to "outfile" instead of the screen.

Note that redirection of input in ASM51 is redundant since the assembler is an absolute assembler and has no command line options other than the file name argument.

Output redirection is useful for speeding up the assembly process. Because assembly-time errors are directed to std_err in DOS, an error listing cannot be redirected to a file.

To make the .lst file serve as an error-only file, use the Cross Assembler Controls $PRINT (create a list file) $NOLIST (turn the listing off). Use the Cross Assembler Controls $NOSYMBOLS to further compress the error-only listing resulting from the manipulation of the list file controls. See Chapter 6 for more information. The errors will be listed in the .lst file, as usual.

If the control $NOPRINT (see Chapter 6) is active, all error messages are send to the screen.

3.6 References

1. IBM Corp., Disk Operating System, Version 1.10, May 1982.
2. IBM Corp., Disk Operating System, Version 2.00, January 1983.

CHAPTER 4

4 8051 INSTRUCTION SET

4.1 Notation

Below is an explanation of the column headings and column contents of the 8051 Instruction Set Summary Table that follows in this chapter.

MNEMONIC

The MNEMONIC column contains the 8051 Instruction Set Mnemonic and a brief description of the instruction's operation.

OPERATION

The OPERATION column describes the 8051 Instruction Set in unambiguous symbology. Following are the definitions of the symbols used in this column.

<n:m>	Bits of a register inclusive. For example, PC<10:0> means bits 0 through 10 inclusive of the PC. Bit 0 is always the least significant bit.
+	Binary addition
-	Binary 2s complement subtraction
/	Unsigned integer division
X	Unsigned integer multiplication
~	Binary complement (1s complement)
^	Logical And
V	Inclusive Or
V	Exclusive Or
>	Greater than
<>	Not equal to
=	Equals
->	Is written into. For example, A + SOper -> A means the result of the binary addition between A and the Source Operand is written into A.
A	The 8-bit Accumulator Register.
AC	The Auxiliary Carry Flag in the Program Status Word
CF	The Carry Flag in the Program Status Word
DOper	The Destination Operand used in the instruction.
DPTR	16-bit Data Pointer
Intrupt Active Flag	Internal Flag that holds off interrupts until the Flag is cleared.
Jump Relative to PC	A Jump that can range between -128 bytes and +127 bytes from the PC value of the next instruction.
Paddr	A 16-bit Program Memory address
PC	The 8051 Program Counter. This 16-bit register points to the byte in the Program Memory space that is fetched as part of the instruction stream.
PM(addr)	Byte in Program Memory space pointed to by addr.
Remainder	Integer remainder of unsigned integer division

Soper	The Source Operand used in the instruction.
SP	8-bit Stack Pointer
STACK	The Last In First Out data structure that is controlled by the 8-bit Stack Pointer (SP). Sixteen bit quantities are pushed on the stack low byte first.

DEST ADDR MODE/SOURCE ADDR MODE

These two columns specify the Destination and Source Addressing Modes, respectively, that are available for each instruction.

AB	The Accumulator-B Register pair.
Accumulator	Operand resides in the accumulator
Bit Direct	Operand is the state of the bit specified by the Bit Memory address.
Carry Flag	Operand is the state of the 1-bit Carry flag in the Program Status Word (PSW).
Data Pointer	Operand resides in the 16-bit Data Pointer Register.
Direct	Operand is the contents of the specified 8-bit Internal Data Memory address from 0 (00H) to 127 (7FH) or a Special Function Register address.
Indirect	Operand is the contents of the address contained in the register specified.
Immediate	Operand is the next sequential byte after the instruction in Program Memory space
Prog Direct	16-bit address in Program Memory Space.
Prog Indir	Operand in Program Memory Space is the address contained in the register specified.
Register	Operand is the contents of the register specified.
Stack	Operand is on the top of the Stack.

ASSEMBLY LANGUAGE FORM

This column contains the correct format of the instructions that are recognized by the Cross Assembler.

A	Accumulator
AB	Accumulator-B Register pair.
C	Carry Flag
Baddr	Bit Memory Direct Address.
Daddr	Internal Data Memory or Special Function Register Direct Address.
Data	8-bit constant data.
data16	16-bit constant data.
DPTR	16-bit Data Pointer Register.
PC	16-bit Program Counter.
Paddr	16-bit Program Memory address
Ri	Indirect Register. R0 or R1 are the only indirect registers.
Roff	8-bit offset for Relative Jump.
Rn	Implicit Register. Each register bank has 8 general purpose registers, designated R0, R1, R2, R3, R4, R5, R6, R7.

HEX OPCODE

This column gives the machine language hexadecimal opcode for each 8051 instruction.

BYT

This column gives the number of bytes in each 8051 instruction.

CYC

This column gives the number of cycles of each 8051 instruction. The time value of a cycle is defined as 12 divided by the oscillator frequency. For example, if running an 8051 family component at 12 MHz, each cycle takes 1 microsecond.

PSW

This column identifies which condition code flags are affected by the operation of the individual instructions. The condition code flags available on the 8051 are the Carry Flag, CF, the Auxiliary Carry Flag, AC, and the Overflow Flag, OV.

It should be noted that the PSW is both byte and bit directly addressable. Should the PSW be the operand of an instruction that modifies it, the condition codes could be changed even if this column states that the instruction doesn't affect them.

0	Condition code is cleared
1	Condition code is set
*	Condition code is modified by instruction
-	Condition code is not affected by instruction

4.2 8051 Instruction Set Summary

MNEMONIC	OPERATION	DEST ADDR MODE	SOURCE ADDR MODE	ASSEMBLY LANGUAGE FORM	HEX OPCODE	BYT	CYC	PSW CF AC OV
ACALL 2K in Page (11 bits) Absolute Call	PC + 2 -> STACK SP + 2 -> SP Paddr<10:0> -> PC<10:0> PC<15:11> -> PC<15:11>	Prog Direct		ACALL Paddr	see note 1	2	2	- - -
ADD Add Operand to Accumulator	A + SOper -> A	Accumulator	Immediate	ADD A,#data	24	2	1	* * *
		"	Direct	ADD A,Daddr	25	2	1	
		"	Indirect	ADD A,@Ri	26,27	1	1	
		"	Register	ADD A,Rn	28-2F	1	1	
ADDC Add Operand with Carry to Accumulator	A + SOper + C -> A	Accumulator	Immediate	ADDC A,#data	34	2	1	* * *
		"	Direct	ADDC A,Daddr	35	2	1	
		"	Indirect	ADDC A,@Ri	36,37	1	1	
		"	Register	ADDC A,Rn	38-3F	1	1	
AJMP 2K in Page (11 bits) Absolute Jump	Paddr<10:0> -> PC<10:0> PC<15:11> -> PC<15:11>	Prog Direct		AJMP Paddr	see note 2	2	2	- - -
ANL Logical AND of Source Operand with Destination Operand	SOper ^ DOper -> DOper	Direct	Accumulator	ANL Daddr,A	52	2	1	- - -
		"	Immediate	ANL Daddr,#data	53	3	2	
		Accumulator	Immediate	ANL A,#data	54	2	1	
		"	Direct	ANL A,Daddr	55	2	1	
		"	Indirect	ANL A,@Ri	56,57	1	1	
		"	Register	ANL A,Rn	58-5F	1	1	
Logical AND of Source Operand with Carry Flag	SOper ^ CF -> CF	Carry Flag	Bit Direct	ANL C,Baddr	82	2	2	* - -

MNEMONIC	OPERATION	DEST ADDR MODE	SOURCE ADDR MODE	ASSEMBLY LANGUAGE FORM	HEX OPCODE	BYT	CYC	PSW CF AC OV
Logical AND of Source Operand Complemented With Carry Flag	~SOper ^ CF -> CF	Carry Flag	Bit Direct	ANL C,/Baddr	B0	2	2	* - -
CJNE Compare Operands and Jump Relative if not Equal	Jump Relative to PC if DOper <> SOper	Accumulator " Indirect Register	Immediate Direct Immediate "	CJNE A,#data,Roff CJNE A,Daddr,Roff CJNE @Ri,#data,Roff CJNE Rn,#data,Roff	B4 B5 B6,B7 B8-BF	3 3 3 3	2 2 2 2	* - - see note 2
CLR Clear Accumulator	0 -> A	Accumulator		CLR A	E4	1	1	- - -
Clear Carry Flag	0 -> CF	Carry Flag		CLR C	C3	1	1	0 - -
Clear Bit Operand	0 -> DOper	Bit Direct		CLR Baddr	C2	2	1	- - -
CPL Complement Accumulator	~A -> A	Accumulator		CPL A	F4	1	1	- - -
Complement Carry Flag	~CF -> CF	Carry Flag		CPL C	B3	1	1	* - -
Complement Bit Operand	~DOper -> DOper	Bit Direct		CPL Baddr	B2	2	1	- - -

DA
Decimal Adjust Accumulator for Addition
	If (A<3:0> > 9) v AC
	then A<3:0>+6 -> A<3:0>
	If (A<7:4> > 9) v CF
	then A<7:4>+6 -> A<7:4>
		Accumulator		DA A	D4	1	1	* - -
								see note 4

DEC Decrement Operand	DOper - 1 -> DOper	Accumulator Direct Indirect Register		DEC A DEC Daddr DEC @Ri DEC Rn	14 15 16,17 16,17	1 2 1 1	1 1 1 1	- - -
DIV Divide Accumulator by B Register	A / B -> A Remainder -> B	AB		DIV AB	84	1	4	0 - * see note 5
DJNZ Decrement Operand and Jump Relative if Not Zero	DOper - 1 -> DOper If DOper <> 0 then Jump Relative to PC	Direct Register		DJNZ Daddr,Roff DJNZ Rn,Roff	D5 D8-DF	3 2	2 2	- - -
INC Increment Operand	DOper + 1 -> DOper	Accumulator Direct Indirect Register Data Pointer		INC A INC Daddr INC @Ri INC Rn INC DPTR	04 05 06,07 08-0F A3	1 2 1 1 1	1 2 1 1 2	- - -
JB Jump Relative if Bit Operand is Set	If DOper = 1 then Jump Relative to PC	Bit Direct		JB Baddr,Roff	20	3	2	- - -
JBC Jump Relative if Bit Operand is Set and Clear Bit Operand	If DOper = 1 then Bit Direct 0 -> DOper and Jump Relative to PC	Bit Direct		JBC Baddr,Roff	10	3	2	* * * see note 6
JC Jump Relative if Carry Flag is Set	If CF = 1 then Jump Relative to PC	Carry Flag		JC Roff	40	2	2	- - -
JMP Jump Indirect	DPTR<15:0> + A<7:0> -> PC<15:0>	Prog Indir		JMP @A+DPTR	73	1	2	- - -
JNB Jump Relative if Bit Operand is Clear	If DOper = 0 then Jump Relative to PC	Bit Direct		JNB Baddr,Roff	30	3	2	- - -
JNC Jump Relative if Carry Flag is Clear	If CF = 0 then Jump Relative to PC	Carry Flag		JNC Roff	50	2	2	- - -
JNZ Jump Relative if the Accumulator is Not Zero	If A<7:0> <> 0 then Jump Relative to PC	Accumulator		JNZ Roff	70	2	2	- - -
JZ Jump Relative if the Accumulator is Zero	If A<7:0> = 0 then Jump Relative to PC	Accumulator		JZ Roff	60	2	2	- - -
LCALL Long (16 bits) Call	PC + 3 -> STACK SP + 2 -> SP Paddr<15:0> -> PC<15:0>	Prog Direct		LCALL Paddr	12	3	2	- - -
LJMP Long (16 bits) Absolute Jump	Paddr<15:0> -> PC<15:0>	Prog Direct		LJMP Paddr	02	3	2	- - -

MNEMONIC	OPERATION	DEST ADDR MODE	SOURCE ADDR MODE	ASSEMBLY LANGUAGE FORM	HEX OPCODE	BYT	CYC	PSW CF AC OV
MOV Move Source Operand To Destination Operand	SOper -> Doper	Accumulator	Immediate	MOV A,#data	74	2	1	- - -
		"	Direct	MOV A,Daddr	E5	2	1	
		"	Indirect	MOV A,@Ri	E6,E7	1	1	
		"	Register	MOV A,Rn	E8-EF	1	1	
		Direct	Accumulator	MOV Daddr,A	F5	2	1	
		"	Immediate	MOV Daddr,#data	75	3	2	
		"	Direct	MOV Daddr,Daddr	85	2	2	
		"	Indirect	MOV Daddr,@Ri	86,87	2	2	
		Indirect	Register	MOV Daddr,Rn	88-8F	2	2	
	SOper -> DOper	"	Accumulator	MOV @Ri,A	F6,F7	1	1	
		"	Immediate	MOV @Ri,#data	76,77	2	1	
		"	Direct	MOV @Ri,Daddr	A6,A7	2	2	
		Register	Accumulator	MOV Rn,A	F8-FF	1	1	
		"	Immediate	MOV Rn,#data	78-7F	2	1	
		"	Direct	MOV Rn,Daddr	A8-AF	2	2	
		Data Pointer	Immediate	MOV DPTR,#data16	90	3	2	
Move Carry Flag to Bit Destination Operand	CF -> Doper	Bit Direct	Carry Flag	MOV Baddr,C	92	2	2	- - -
Move Bit Destination Operand to Carry Flag	DOper -> CF	Carry Flag	Bit Direct	MOV C,Baddr	A2	2	1	* - -
MOVC Move byte from Program Memory to the Accumulator	PM(DPTR<15:0> + A<7:0>) -> A<7:0>	Accumulator	Prog Ind	MOVC A,@A+DPTR	93	1	2	- - -
	PM(PC<15:0> + A<7:0>) -> A<7:0>	Accumulator	Prog Ind	MOVC A,@A+PC	83	1	2	- - -
MOVX Move byte from External Data Memory to the Accumulator	SOper -> A	Accumulator	Indirect	MOVX A,@Ri	E2,E3	1	2	- - -
		"	"	MOVX A,@DPTR	E0	1	2	
Move byte in the Accumulator to External Data Memory	A -> DOper	Indirect	Accumulator	MOVX @Ri,A	F2,F3	1	2	- - -
		"	"	MOVX @DPTR,A	F0	1	2	
MUL Multiply Accumulator by B Register	A X B -> B,A (see note 7)	AB		MUL AB	A4	1	4	0 - *
NOP No Operation				NOP	00	1	1	- - -
ORL Logical Inclusive OR of Source Operand with Destination Operand	SOper v DOper -> DOper	Direct	Accumulator	ORL Daddr,A	42	2	1	- - -
		"	Immediate	ORL Daddr,#data	43	3	2	
		Accumulator	Immediate	ORL A,#data	44	2	1	
		"	Direct	ORL A,Daddr	45	2	1	
		"	Indirect	ORL A,@Ri	46,47	1	1	
		"	Register	ORL A,Rn	48-4F	1	1	
Logical Inclusive OR of Source Operand with Carry Flag	SOper v CF -> CF	Carry Flag	Bit Direct	ORL C,Baddr	72	2	2	* - -
Logical Inclusive OR of Source Operand Complemented with Carry Flag	~SOper v CF -> CF	Carry Flag	Bit Direct	ORL C,/Baddr	A0	2	2	* - -
POP Pop Stack and Place in Destination Operand	~SOper v CF -> CF	Direct	Stack	POP Daddr	D0	2	2	- - -
PUSH Push Source Operand onto Stack	SP + 1 -> SP SOper -> STACK	Stack	Direct	PUSH Daddr	C0	2	2	- - -
RET Return from Subroutine	STACK -> PC<15:8> SP - 1 -> SP STACK -> PC<7:0 SP - 1 -> SP			RET	22	1	2	- - -
RETI Return from Interrupt Routine	STACK -> PC<15:8> SP - 1 -> SP STACK -> PC<7:0> SP - 1 -> SP 0 -> Intrupt Active Flag			RETI	32	1	2	- - -
RL Rotate Accumulator Left One Bit	A<6:0> -> A<7:1> A<7> -> A<0>	Accumulator		RL A	23	1	1	- - -
RLC Rotate Accumulator Left One Bit Thru the Carry Flag	A<6:0> -> A<7:1> CF -> A<0> A<7> -> CF	Accumulator		RLC A	33	1	1	* - -
RR Rotate Accumulator Right One Bit	A<7:1> -> A<6:0> A<0> -> A<7>	Accumulator		RR A	03	1	1	- - -
RLC Rotate Accumulator Right One Bit Thru the	A<7:1> -> A6:0> CF -> A<7>	Accumulator		RRC A	13	1	1	* - -

MNEMONIC	OPERATION	DEST ADDR MODE	SOURCE ADDR MODE	ASSEMBLY LANGUAGE FORM	HEX OPCODE	BYT	CYC	PSW CF AC OV
Carry Flag	A<0> -> CF							
SETB								
Set Bit Operand	1 -> CF	Carry Flag		SETB C	D3	1	1	1 - -
	1 -> DOper	Bit Direct		SETB Baddr	D2	2	1	- - -
SJMP								
Short (8 bits) Relative Jump	Jump Relative to PC			SJMP Roff	80	2	2	- - -
SUBB								
Subtract Operand with Borrow from the Accumulator	A - SOper - CF -> A	Accumulator	Immediate	SUBB A,#data	94	2	1	* * *
		"	Direct	SUBB A,Daddr	95	2	1	
		"	Indirect	SUBB A,@Ri	96,97	1	1	
		"	Register	SUBB A,Rn	98-9F	1	1	
SWAP								
Swap Nibbles within the Accumulator	A<7:4> -> A<3:0>	Accumulator		SWAP A	C4	1	1	- - -
	A<3:0> -> A<7:4>							
XCH								
Exchange bytes of the Accumulator and the Source Operand	SOper<7:0> -> A<7:0>	Accumulator	Direct	XCH A,Daddr	C5	2	1	- - -
	A<7:0> -> SOper<7:0>	"	Indirect	XCH A,@Ri	C6,C7	1	1	
		"	Register	XCH A,Rn	C8-Cf	1	1	
XCHD								
Exchange the Least Significant Nibble of the Accumulator and the Source Operand	SOper<3:0> -> A<3:0>	Accumulator	Indirect	XCHD A,@Ri	D6,D7	1	1	- - -
	A<3:0> -> SOper<3:0>							
XRL								
Logical Exclusive OR of Source Operand with Destination Operand	SOper v DOper -> DOper	Direct	Accumulator	XRL Daddr,A	62	2	1	- - -
		"	Immediate	XRL Daddr,#data	63	3	2	
		Accumulator	Immediate	XRL A,#data	64	2	1	
		"	Direct	XRL A,Daddr	65	2	1	
		"	Indirect	XRL A,@Ri	66,67	1	1	
		"	Register	XRL A,Rn	68-6F	1	1	

4.3 Notes

1 There are 8 possible opcodes. Starting with 11H as the opcode base, the final opcode is formed by placing bits 8, 9 and 10 of the target address in bits 5, 6 and 7 of the opcode. The 8 possible opcodes in hexadecimal are then: 11, 31, 51, 71, 91, B1, D1, F1.

2 There are 8 possible opcodes. Starting with 01H as the opcode base, the final opcode is formed by placing bits 8, 9 and 10 of the target address in bits 5, 6 and 7 of the opcode. The 8 possible opcodes in hexadecimal are then: 01, 21, 41, 61, 81, A1, C1, E1.

3 The Carry Flag is set if the Destination Operand is less than the Source Operand. Otherwise the Carry Flag is cleared.

4 The Carry Flag is set if the BCD result in the Accumulator is greater than decimal 99.

5 The Overflow Flag is set if the B Register contains zero (flags a divide by zero operation). Otherwise the Overflow Flag is cleared.

6 If any of the condition code flags are specified as the operand of this instruction, they will be reset by the instruction if they were originally set.

7 The high byte of the 16-bit product is placed in the B Register, the low byte in Accumulator.

4.4 References

1. Intel Corp., Microcontroller Handbook, 1984.

CHAPTER 5

5 8051 CROSS ASSEMBLER DIRECTIVES

5.1 Introduction

The 8051 Cross Assembler Directives are used to define symbols, reserve memory space, store values in program memory, select various memory spaces, set the current segment's location counter and identify the end of the source file.

Only one directive per line is allowed, however comments may be included. The remaining part of this chapter details the function of each directive.

5.2 Symbol Definition Directives

EQU Directive

The EQUate directive is used to assign a value to a symbol. It can also be used to specify user defined names for the implicit operand symbols predefined for the Accumulator (i.e., A) and the eight General Purpose Registers (i.e., R0 thru R7).

The format for the EQU directive is: symbol, followed by one or more spaces or tabs, followed by EQU, followed by one or more spaces or tabs, followed by a number, arithmetic expression, previously defined symbol (no forward references allowed) or one of the allowed implicit operand symbols (e.g., A, R0, R1, R2, R3, R4, R5, R6, R7), followed by an optional comment.

Below are examples of using the EQU Directive:

```
TEN         EQU   10          ;Symbol equated to a number
COUNTER     EQU   R7          ;User defined symbol for the implicit
                              ;operand symbol R7.  COUNTER can now
                              ;be used wherever it is legal to use
                              ;R7.  For example the instruction
                              ;INC R7 could now be written INC COUNTER.
ALSO_TEN    EQU   TEN         ;Symbol equated to a previously defined
                              ;symbol.
FIVE        EQU   TEN/2       ;Symbol equated to an arithmetic exp.
A_REG       EQU   A           ;User defined symbol for the implicit
                              ;operand symbol A.
ASCII_D     EQU   'D'         ;Symbol equated to an ASCII literal
```

SET Directive

Similar to the EQU directive, the SET directive is used to assign a value or implicit operand to a user defined symbol. The difference however, is that with the EQU directive, a symbol can only be defined once. Any attempt to define the symbol again will cause the Cross Assembler to flag it as an error. On the other hand, with the SET directive, symbols may be redefined. There is no limit to the number of times a symbol can be redefined with the SET directive.

The format for the SET directive is: symbol, followed by one or more spaces or tabs, followed by SET, followed by one or more spaces or tabs, followed by a number, arithmetic expression, previously defined symbol (no forward references allowed) or one of the allowed implicit operand symbols (e.g., A, R0, R1, R2, R3, R4, R5, R6, R7), followed by an optional comment.

Below are examples of using the SET Directive:

```
POINTER     SET   R0          ;Symbol equated to register 0
POINTER     SET   R1          ;POINTER redefined to register 1

COUNTER     SET   1           ;Symbol initialized to 1
COUNTER     SET   COUNTER+1   ;An incrementing symbol
```

BIT Directive

The BIT Directive assigns an internal bit memory direct address to the symbol. If the numeric value of the address is between 0 and 127 decimal, it is a bit address mapped in the Internal Memory Space. If the numeric value of the address is between 128 and 255, it is an address of a bit located in one of the Special Function Registers. Addresses greater than 255 are illegal and will be flagged as an error.

The format for the BIT Directive is: symbol, followed by one or more spaces or tabs, followed by BIT, followed by one or more spaces or tabs, followed by a number, arithmetic expression, or previously defined symbol (no forward references allowed), followed by an optional comment.

Below are examples of using the BIT Directive:

```
CF          BIT   0D7H        ;The single bit Carry Flag in PSW
OFF_FLAG    BIT   6           ;Memory address of single bit flag
ON_FLAG     BIT   OFF_FLAG+1  ;Next bit is another flag
```

CODE Directive

The CODE Directive assigns an address located in the Program Memory Space to the symbol. The numeric value of the address cannot exceed 65535.

The format for the CODE Directive is: symbol, followed by one or more spaces or tabs, followed by CODE, followed by one or more spaces or tabs, followed by a number, arithmetic expression, or previously defined symbol (no forward references allowed), followed by an optional comment.

Below are examples of using the CODE Directive:

```
RESET       CODE  0
EXTI0       CODE  RESET + (1024/16)
```

DATA Directive

The DATA Directive assigns a directly addressable internal memory address to the symbol. If the numeric value of the address is between 0 and 127 decimal, it is an address of an Internal Data Memory location. If the numeric value of the address is between 128 and 255, it is an address of a Special Function Register. Addresses greater than 255 are illegal and will be flagged as an error.

The format for the DATA Directive is: symbol, followed by one or more spaces or tabs, followed by DATA, followed by one or more spaces or tabs, followed by a number, arithmetic expression, or previously defined symbol (no forward references allowed), followed by an optional comment.

Below are examples of using the DATA Directive:

```
PSW         DATA  0D0H        ;Defining the Program Status address
BUFFER      DATA  32          ;Internal Data Memory address
FREE_SPAC   DATA  BUFFER+16   ;Arithmetic expression.
```

IDATA Directive

The IDATA Directive assigns an indirectly addressable internal data memory address to the symbol. The numeric value of the address can be between 0 and 255 decimal. Addresses greater than 255 are illegal and will be flagged as an error.

The format for the IDATA Directive is: symbol, followed by one or more spaces or tabs, followed by IDATA, followed by one or more spaces or tabs, followed by a number, arithmetic expression, or previously defined symbol (no forward references allowed), followed by an optional comment.

Below are examples of using the IDATA Directive:

```
TOKEN       IDATA 60
BYTE_CNT    IDATA TOKEN + 1
ADDR        IDATA TOKEN + 2
```

XDATA Directive

The XDATA Directive assigns an address located in the External Data Memory Space to the symbol. The numeric value of the address cannot exceed 65535.

The format for the XDATA Directive is: symbol, followed by one or more spaces or tabs, followed by XDATA, followed by one or more spaces or tabs, followed by a number, arithmetic expression, or previously defined symbol (no forward references allowed), followed by an optional comment.

Below are examples of using the XDATA Directive:

```
USER_BASE   XDATA 2048
HOST_BASE   XDATA USER_BASE + 1000H
```

5.3 Segment Selection Directives

There are five Segment Selection Directives: CSEG, BSEG, DSEG, ISEG, XSEG, one for each of the five memory spaces in the 8051 architecture. The CSEG Directive is used to select the Program Memory Space. The BSEG Directive is used to select the Bit Memory Space. The DSEG Directive is used to select the directly addressable Internal Data Memory Space. The ISEG is used to select the indirectly addressable Internal Data Memory Space. The XSEG is used to select the External Data Memory Space.

Each segment has its own location counter that is reset to zero during the Cross Assembler program initialization. The contents of the location counter can be overridden by using the optional AT after selecting the segment.

The Program Memory Space, or CSEG, is the default segment and is selected when the Cross Assembler is run.

The format of the Segment Selection Directives are: zero or more spaces or tabs, followed by the Segment Selection Directive, followed by one or more spaces or

tabs, followed by the optional segment location counter override AT command and value, followed by an optional comment.

The value of the AT command can be a number, arithmetic expression or previously defined symbol (forward references are not allowed). Care should be taken to ensure that the location counter does not advance beyond the limit of the selected segment.

Below are examples of the Segment Selection Directives:

```
DSEG                    ;Select direct data segment using
                        ;current location counter value.
BSEG AT 32              ;Select bit data segment forcing
                        ;location counter to 32 decimal.
XSEG AT (USER_BASE * 5) MOD 16 ;Arithmetic expressions can be
                        ;used to specify location.
```

5.4 Memory Reservation and Storage Directives

DS Directive

The DS Directive is used to reserve space in the currently selected segment in byte units. It can only be used when ISEG, DSEG or XSEG are the currently active segments. The location counter of the segment is advanced by the value of the directive. Care should be taken to ensure that the location counter does not advance beyond the limit of the segment.

The format for the DS Directive is: optional label, followed by one or more spaces or tabs, followed by DS, followed by one or more spaces or tabs, followed by a number, arithmetic expression, or previously defined symbol (no forward references allowed), followed by an optional comment.

Below is an example of using the DS Directive in the internal Data Segment. If, for example, the Data Segment location counter contained 48 decimal before the example below, it would contain 104 decimal after processing the example.

```
DSEG                          ;Select the data segment
DS          32                ;Label is optional
SP_BUFFER:  DS    16          ;Reserve a buffer for the serial port
IO_BUFFER:  DS    8           ;Reserve a buffer for the I/O
```

DBIT Directive

The DBIT Directive is used to reserve bits within the BIT segment. It can only be used when BSEG is the active segment. The location counter of the segment is advanced by the value of the directive. Care should be taken to ensure that the location counter does not advance beyond the limit of the segment.

The format for the DBIT Directive is: optional label, followed by one or more spaces or tabs, followed by DBIT, followed by one or more spaces or tabs, followed by a number, arithmetic expression, or previously defined symbol (no forward references allowed), followed by an optional comment.

Below is an example of using the DBIT Directive:

```
BSEG                          ;Select the bit segment
DBIT        16                ;Label is optional
IO_MAP:     DBIT  32          ;Reserve a bit buffer for I/O
```

DB Directive

The DB Directive is used to store byte constants in the Program Memory Space. It can only be used when CSEG is the active segment.

The format for the DB Directive is: optional label, followed by one or more spaces or tabs, followed by DB, followed by one or more spaces or tabs, followed by the byte constants that are separated by commas, followed by an optional comment.

The byte constants can be numbers, arithmetic expressions, symbol values or ASCII literals. ASCII literals have to be delimited by apostrophes ('), but they can be strung together up to the length of the line.

Below are examples of using the DB Directive. If an optional label is used, its value will point to the first byte constant listed.

```
COPYRGHT_MSG:
            DB    '(c) Copyright, 1984'   ;ASCII Literal
RUNTIME_CONSTANTS:
            DB    127,13,54,0,99          ;Table of constants
            DB    17,32,239,163,49        ;Label is optional
            MIXED: DB 2*8,'MPG',2*16,'abc' ;Can mix literals & no.
```

DW Directive

The DW Directive is used to store word constants in the Program Memory Space. It can only be used when CSEG is the active segment.

The format for the DW Directive is: optional label, followed by one or more spaces or tabs, followed by DW, followed by one or more spaces or tabs, followed by the word constants that are separated by commas, followed by an optional comment.

The word constants can be numbers, arithmetic expressions, symbol values or ASCII literals. ASCII literals must be delimited by apostrophes ('), but unlike the DB Directive, only a maximum of two ASCII characters can be strung together. The first character is placed in the high byte of the word and the second character is placed in the low byte. If only one character is enclosed by the apostrophes, a zero will be placed in the high byte of the word.

Below are examples of using the DW Directive. If an optional label is used, its value will point to the high byte of the first word constant listed.

```
JUMP_TABLE: DW    RESET,START,END   ;Table of addresses
            DW    TEST,TRUE,FALSE   ;Optional label
RADIX:      DW    'H',1000H         ;1st byte contains 0
                                    ;2nd byte contains 48H (H)
                                    ;3rd byte contains 10H
                                    ;4th byte contains 0
```

5.5 Miscellaneous Directives

ORG Directive

The ORG Directive is used to specify a value for the currently active segment's location counter. It cannot be used to select segments like the directives above. It can only be used within a segment when the location counter needs to be changed.

Care should be taken to ensure that the location counter does not advance beyond the limit of the selected segment.

The format of the ORG Directive is: zero or more spaces or tabs, followed by ORG, followed by one or more spaces or tabs, followed by a number, arithmetic expression, or previously defined symbol (no forward references are allowed), followed by an optional comment.

Below are examples of the ORG directive.

```
      ORG  1000H                ;Location counter set at 4096 decimal
      ORG  RESET                ;Previously defined symbol
      ORG  BASE + MODULE_NO     ;Arithmetic expression
```

USING Directive

The USING Directive is used to specify which of the four General Purpose Register banks is used in the code that follows the directive. It allows the use of the predefined register symbols AR0 through AR7 instead of the register's direct addresses. It should be noted that the actual register bank switching must still be done in the code. This directive simplifies the direct addressing of a specified register bank.

The format of the USING Directive is: zero or more spaces or tabs, followed by USING, followed by one or more spaces or tabs, followed by a number, arithmetic expression, or previously defined symbol (no forward references are allowed), followed by an optional comment.

The number, arithmetic expression, or previously defined symbol must result in a number between 0 and 3 in order to specify one of the four register banks in the 8051.

The following table maps the specified value in the USING directive with the direct addresses of the predefined symbols.

Predefined Symbol	USING Value			
	0	1	2	3
AR0	0	8	16	24
AR1	1	9	17	25
AR2	2	10	18	26
AR3	3	11	19	27
AR4	4	12	20	28
AR5	5	13	21	29
AR6	6	14	22	30
AR7	7	15	23	31

Below are examples of the USING Directive:

```
USING       0                 ;Select addresses for Bank 0
USING       1+1+1             ;Arithmetic expressions
```

END Directive

The END Directive is used to signal the end of the source program to the Cross Assembler. Every source program must have one and only one END Directive. A missing END Directive, as well as text beyond the occurrence of the END Directive are not allowed and will be flagged as errors.

The format of the END Directive is: zero or more spaces or tabs, followed by END, followed by an optional comment. All text must appear in the source program before the occurrence of the END Directive.

Below is an example of the END Directive:

```
            END         ;This is the End
```

5.6 Conditional Assembly Directives

IF, ELSE and ENDIF Directive

The IF, ELSE and ENDIF directives are used to define conditional assembly blocks. A conditional assembly block begins with an IF statement and must end with the ENDIF directive. In between the IF statement and ENDIF directive can be any number of assembly language statements, including directives, controls, instructions, the ELSE directive and nested IF-ENDIF conditional assembly blocks.

The IF statement starts with the keyword IF, followed by one or more spaces or tabs, followed by a number, arithmetic expression, or previously defined symbol (no forward references are allowed), followed by an optional comment. The number, arithmetic expression or symbol is evaluated and if found to be TRUE (nonzero), the assembly language statements are translated up to the next ELSE or ENDIF directives. If the IF statement was evaluated FALSE (zero), the assembly language statements are considered null up to the next ELSE or ENDIF directives.

If an optional ELSE appears in the conditional assembly block, the assembly language statements following are handled oppositely from the assembly language statements following the IF statement. In other words, if the IF statement was evaluated TRUE, the statements following it are translated, while the statements following the ELSE will be handled as if they were null. On the other hand, if the IF statement was evaluated FALSE, only the assembly language statements following the ELSE directive would be translated.

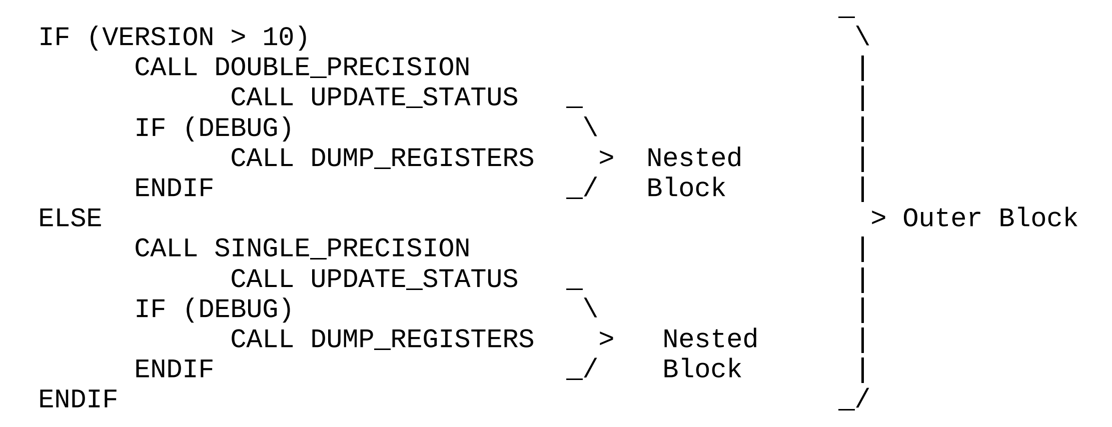
IF-ELSE-ENDIF conditional assembly blocks can be nested up to 255 levels deep. The following are some examples of conditional assembly blocks. This first conditional assembly block simply checks the symbol DEBUG. If DEBUG is non-zero, the MOV and CALL instructions will be translated by the Cross Assembler.

```
      IF (DEBUG)
            MOV    A,#25
            CALL   OUTPUT
      ENDIF
```

The next example used the optional ELSE directive. If SMALL_MODEL is zero, only the statements following the ELSE directive will be translated.

```
      IF (SMALL_MODEL)
            MOV  R0,#BUFFER
            MOV  A,@R0
      ELSE
            MOV  R0,#EXT_BUFFER
            MOVX A,@R0
      ENDIF
```

The last example shows nested conditional assembly blocks. Conditional assembly blocks can be nested up to 255 levels deep. Every level of nesting must have balanced IF-ENDIF statements.



```
                                                    _
IF (VERSION > 10)                                    \
      CALL DOUBLE_PRECISION                           |
            CALL UPDATE_STATUS   _                    |
      IF (DEBUG)                  \                   |
            CALL DUMP_REGISTERS    >  Nested          |
      ENDIF                      _/   Block           |
ELSE                                                   > Outer Block
      CALL SINGLE_PRECISION                           |
            CALL UPDATE_STATUS   _                    |
      IF (DEBUG)                  \                   |
            CALL DUMP_REGISTERS    >   Nested         |
      ENDIF                      _/    Block          |
ENDIF                                               _/
```

CHAPTER 6

6 8051 CROSS ASSEMBLER CONTROLS

6.1 Introduction

Assembler controls are used to control where the Cross Assembler gets its input source file, where it stores the object file, how it formats and where it outputs the listing.

All Assembler controls are prefaced with a dollar sign, ($) . No spaces or tabs are allowed between the dollar sign and the body of the control. Also, only one control per line is permitted. Comments are allowed on the same line as an Assembler control.

There are two types of controls, Primary controls and General controls. Primary controls can be invoked only once per assembly. If an attempt is made to change a previously invoked primary control, the attempt is ignored. For example, if $NOPRINT is put on line 1 of the source file and $PRINT is put on line 2, the $PRINT control will be ignored and the listing will not be output. General controls can be invoked any number of times in a source program.

There are two legal forms for each Assembler control, the full form and the abbreviated form. The two forms can be used interchangeable in the source program.

Below is a description of each Assembler control. Assembler controls with common functionality are grouped together.

6.2 Assembler Control Descriptions

$DATE(date)

Places the ASCII string enclosed by parenthesis in the date field of the page header. The ASCII string can be from 0 to 9 characters long.

```
      CONTROL:    $DATE(date)
      ABBREV:     $DA(date)
      TYPE:       Primary
      DEFAULT:    No date in page header
      EXAMPLES:   $DATE(1-JUL-84)
                  $DA(7/22/84)
```

$DEBUG(file)
$NODEBUG

These controls determine whether or not a MetaLink Absolute Object Module format file is created. The MetaLink Absolute Object Module format file is used in conjunction with MetaLink's MetaICE series of in-circuit-emulators. Among other advantages, it provides powerful symbolic debug capability in the emulator debug environment. $NODEBUG
specifies that a MetaLink Absolute Object Module file will not be created. $DEBUG specifies that a MetaLink Absolute Object Module file will be created. The $DEBUG control allows any legal file name to be specified as the MetaLink Absolute Object Module filename. If no filename is specified, a default name is used. The default name used for the file is the source file name root with a .DBG extension. If the $DEBUG control is used, both a MetaLink Absolute Object Module file and a standard

Intel Hexadecimal format object file can be generated at the same time. Refer to the $OBJECT control description later in this chapter for information on controlling the Hexadecimal format object file output.

```
      CONTROL:    $DEBUG(file)
                  $NODEBUG
      ABBREV:     $DB(file)
                  $NODB
      DEFAULT:    $NODEBUG
      TYPE:       Primary
      EXAMPLES:   $DB(A:NEWNAME.ICE)
                  $DEBUG
                  $NOOBJECT
```

$EJECT

Places a form feed (ASCII 0CH) in the listing output. The $NOPAGING control will override this control.

```
      CONTROL:    $EJECT
      ABBREV:     $EJ
      DEFAULT:    No form feeds in listing output
      TYPE:       General
      EXAMPLES:   $EJECT
                  $EJ
```

$INCLUDE(file)

Inserts a file in source program as part of the input source program. The file field in this control can be any legal file designator. No extension is assumed, so the whole file name must be specified. Any number of files can be included in a source program. Includes can be nested up to 8 level deep. It is important to note that this control inserts files, it does not chain or concatenate files.

```
      CONTROL:    $INCLUDE(file)
      ABBREV:     $IC(file)
      DEFAULT:    No file included in source program
      TYPE:       General
      EXAMPLES:   $INCLUDE(B:COMMON.EQU
                  $IC(TABLES.ASM)         ;Uses default drive
```

$LIST
$NOLIST

These controls determine whether or not the source program listing is output or not. $LIST will allow the source program listing to be output. $NOLIST stops the source program listing from being output. The $NOPRINT control overrides the $LIST control.

```
      CONTROL:    $LIST
      $NOLIST
      ABBREV:     $LI
                  $NOLI
      DEFAULT:    $LIST
      TYPE:       General
      EXAMPLES:   $NOLIST                 ;This will cause the included
                  $INCLUDE(COMMON.TBL)    ;file not to be listed
                  $LI                     ;Listing continues
```

```
$MOD51
$MOD52
$MOD44
$MOD515
$MOD512
$MOD517
$MOD152
$MOD451
$MOD452
$MOD751
$MOD752
$MOD154
$MOD252
$MOD521
$MOD552
$MOD652
$MOD851
$NOMOD
```

Recognizes predefined special function register symbols in the source program. This saves the user from having to define all the registers in the source program. Appendix B lists the symbols that are defined by these controls. $NOMOD disables the recognizing function. These controls access files of the same name that are included with the MetaLink 8051 CROSS ASSEMBLER distribution diskette. When a $MOD control is used in a source program, it is important that the $MOD file be available to the Cross Assembler. The Cross Assembler first looks for the $MOD file on the default drive, if it isn't found there, the Cross Assembler looks for it on the A: drive. The components supported by each switch are:

```
$MOD51:  8051, 8751, 8031, 80C51, 80C31, 87C51, 9761, 8053
$MOD52:  8052, 8032, 8752
$MOD44:  8044, 8344, 8744
$MOD515: 80515, 80535, 80C515, 80C535
$MOD512: 80512, 80532
$MOD517: 80C517, 80C537
$MOD152: 80C152, 83C152, 80C157
$MOD451: 80C451. 83C451, 87C451
$MOD452: 80C452, 83C452, 87C452
$MOD752: 83C752, 87C752
$MOD751: 83C751, 87C751
$MOD154: 83C514, 80C154, 85C154
$MOD252: 80C252, 83C252, 87C252, 80C51FA, 83C51FA, 87C51FA,
         83C51FB, 87C51FB
$MOD521: 80C521, 80C321, 87C521, 80C541, 87C541
$MOD552: 80C552, 83C552, 87C552
$MOD652: 80C652, 83C652
$MOD851: 80C851, 83C851
```

```
CONTROL:    $MOD51
            $MOD52
            $MOD44
            $MOD152
            $MOD515
            $MOD512
            $MOD451
            $MOD452
            $MOD751
            $MOD752
            $MOD154
            $MOD252
            $MOD521
```

```
                  $MOD552
                  $MOD652
                  $MOD517
                  $MOD851
                  $NOMOD
      ABBREV:
      DEFAULT:    $NOMOD
      TYPE:       Primary
      EXAMPLES:   $MOD51
                  $MOD52
                  $MOD44
                  $MOD152
                  $MOD515
                  $MOD512
                  $MOD451
                  $MOD452
                  $MOD751
                  $MOD752
                  $MOD154
                  $MOD252
                  $MOD521
                  $MOD552
                  $MOD652
                  $MOD517
                  $MOD851
                  $NOMOD
```

$OBJECT(file)
$NOOBJECT

These controls determine whether or not a standard Intel Hexadecimal format object file is created. $NOOBJECT specifies that an object file will not be created. $OBJECT specifies that an object file will be created. If other than the default name is to be used for the object file, the $OBJECT control allows any legal file name to be specified as the object filename. The default name used for the object file is the source file name root with a .HEX extension.

```
      CONTROL:    $OBJECT(file)
      $NOOBJECT
      ABBREV:     $OJ(file)
                  $NOOJ
      DEFAULT:    $OBJECT(source.HEX)
      TYPE:       Primary
      EXAMPLES:   $OJ(A:NEWNAME.OBJ)
                  $NOOBJECT
```

$PAGING
$NOPAGING

These controls specify whether or not the output listing will be broken into pages or will be output as one continuous listing. When the $NOPAGING control is used, the $EJECT and $PAGELENGTH controls are ignored. With the $PAGING control, a form feed and header line is inserted into the output listing whenever an $EJECT control is met, or whenever the number of lines output on the current page exceeds the value specified by the $PAGELENGTH control. The header line contains source file name, title (if $TITLE control was used), date (if $DATE control was used) and page number.

```
      CONTROL:    $PAGING
                  $NOPAGING
```

```
ABBREV:     $PI
            $NOPI
DEFAULT:    $PAGING
TYPE:       Primary
EXAMPLES:   $PAGING
            $NOPI
```

$PAGELENGTH(n)

Sets the maximum number of lines, (n), on a page of the output listing. If the maximum is exceeded, a form feed and page header is inserted in the output listing. This control allows the number of lines per page to be set anywhere between 10 and 255. If the number of lines specified is less than 10, pagelength will be set to 10. If the number of lines specified is greater than 255, pagelength will be set to 255.

The $NOPAGING control will override this control.

```
CONTROL:    $PAGELENGTH(n)
ABBREV:     $PL(n)
DEFAULT:    $PAGELENGTH(60)
TYPE:       Primary
EXAMPLES:   $PAGELENGTH(48)
            $PL(58)
```

$PAGEWIDTH(n)

Sets the maximum number of characters, (n), on a line of the output listing. This control allows the number of characters per line to be set anywhere between 72 and 132. If the number specified is less than 72, the pagewidth is set at 72. If the number specified is greater than 132, the pagewidth is set at 132. If the pagewidth is specified between 72 and 100 and the line being output exceeds the pagewidth specification, the line is truncated at the specified pagewidth and a carriage return/line feed pair is inserted in the listing. If the pagewidth is specified to be greater than 100 and the line being output exceed the pagewidth specification, a carriage return/line feed pair is inserted at the specified pagewidth and the line will continue to be listed on the next line beginning at column 80.

```
CONTROL:    $PAGEWIDTH(n)
ABBREV:     $PW(n)
DEFAULT     $PAGEWIDTH(72)
TYPE:       Primary
EXAMPLES:   $PAGEWIDTH(132)
            $PW(80)
```

$PRINT(file)
$NOPRINT

These controls determine whether or not a listing file is created. $NOPRINT specifies that a listing file will not be created. $PRINT specifies that an listing file will be created. If other than the default name is to be used for the listing file, the $PRINT control allows any legal file name to be specified as the listing filename. The default name used for the listing file is the source file name root with a .LST extension.

```
CONTROL:    $PRINT(file)
            $NOPRINT
ABBREV:     $PR
            $NOPR
DEFAULT:    $PRINT(source.LST)
```

TYPE: Primary
EXAMPLES: $PRINT(A:CONTROL.OUT)
$NOPR

$SYMBOLS
$NOSYMBOLS

Selects whether or not the symbol table is appended to the listing output. $SYMBOLS causes the symbol table to be sorted alphabetically by symbol, formatted and output to the listing file. Along with the symbol name, its value and type are output. Values are output in hexadecimal. Types include NUMB (number), ADDR (address), REG (register symbol) and ACC (accumulator symbol). If a symbol was of type ADDR, it segment is also output as either C (code), D (data) or X (external). Other information listed with the symbols is NOT USED (symbol defined but never referenced), UNDEFINED (symbol referenced but never defined) and REDEFINEABLE (symbol defined using the SET directive). The type and value listed for a REDEFINABLE symbol is that of its last definition in the source program. $NOSYMBOLS does not output the symbol table.

CONTROL: $SYMBOLS
$NOSYMBOLS
ABBREV: $SB
$NOSB
DEFAULT: $SYMBOLS
TYPE: Primary
EXAMPLES: $SB
$NOSYMBOLS

$TITLE(string)

Places the ASCII string enclosed by the parenthesis in the title field of the page header. The ASCII string can be from 0 to 64 characters long. If the string is greater than 64 characters or if the width of the page will not support such a long title, the title will be truncated. If parentheses are part of the string, they must be balanced.

CONTROL: $TITLE(string)
ABBREV: $TT(string)
DEFAULT: No title in page header
TYPE: Primary
EXAMPLES: $TITLE(SAMPLE PROGRAM V1.2)
$TT(METALINK (TM) CROSS ASSEMBLER)

CHAPTER 7

7 8051 CROSS ASSEMBLER MACRO PROCESSOR

7.1 Introduction

Macros are useful for code that is used repetitively throughout the program. It saves the programmer the time and tedium of having to specify the code every time it is used. The code is written only once in the macro definition and it can be used anywhere in the source program any number of times by simply using the macro name.

Sometimes there is confusion between macros and subroutines. Subroutines are common routines that are written once by the programmer and then accessed by CALLing them. Subroutines are usually used for longer and more complex routines where the call/return overhead can be tolerated. Macros are commonly used for simpler routines or where the speed of in-line code is required.

7.2 Macro Definition

Before a macro can be used, it first must be defined. The macro definition specifies a template that is inserted into the source program whenever the macro name is encountered. Macro definitions can not be nested, but once a macro is defined, it can be used in other macro definitions. Macros used this way can be nested up to nine levels deep.

The macro definition has three parts to it: 1) the macro header which specifies the macro name and its parameter list, 2) the macro body which is the part that is actually inserted into the source program, and 3) the macro terminator.

The macro header has the following form:

```
Name        MACRO <parameter list>
```

The name field contains a unique symbol that it used to identify the macro. Whenever that symbol is encountered in the source program, the Cross Assembler will automatically insert the macro body in the source program at that point. The name must be a unique symbol that follows all the rules of symbol formation as outlined in Chapter 2.

The MACRO field of the macro header contains the keyword MACRO. This is used to notify the Cross Assembler that this is the beginning of a macro definition.

The <parameter list> field of the macro header lists anywhere from zero to 16 parameters that are used in the macro body and are defined at assembly time. The symbols used in the parameter list are only used by the Cross Assembler during the storing of the macro definition. As a result, while symbols used in the parameter list must be unique symbols that follow all the rules of symbol formation as outlined in Chapter 2, they can be reissued in the parameter list of another macro definition
without conflict. Parameter list items are separated from one another by a comma. The following are examples of macro definition headers:

```
MULT_BY_16  MACRO                             (no parameters)
DIRECT_ADD  MACRO       DESTINATION,SOURCE      (two parameters)
```

The macro body contains the template that will replace the macro name in the source program. The macro body can contain instructions, directives, conditional assembly statements or controls. As a matter of fact, the macro body can contain any legal Cross Assembler construct as defined in Chapters 2, 4, 5 and 6.

There are two macro definition terminators: ENDM and EXITM. Every macro definition must have an ENDM at the end of its definition to notify the Cross Assembler that the macro definition is complete. The EXITM terminator is an alternative ending of the macro that is useful with conditional assembly statements. When a EXITM is encountered in a program, all remaining statements (to the ENDM) are ignored.

The following is an example of a macro definition that multiplies the Accumulator by 16:

```
MULT_BY_16  MACRO
                  RL   A            ;* 2
                  RL   A            ;* 4
                  RL   A            ;* 8
                  RL   A            ;* 16
            ENDM
```

The following is an example of a macro that adds two numbers together. This could be used by the programmer to do direct memory to memory adds of external variables (create a virtual instruction).

```
DIRECT_ADDX MACRO DESTINATION, SOURCE     (two parameters)
            MOV   R0,#SOURCE
            MOVX  A,@R0
            MOV   R1,A
            MOV   R0,#DESTINATION
            MOVX  A,@R0
            ADD   A,R1
            MOVX  @R0,A
            ENDM
```

A final macro definition example shows the use of the EXITM macro terminator. If CMOS is non-zero, the MOV and only the MOV instruction will be translated by the Cross Assembler.

```
      IDLE  MACRO
               IF (CMOS)
                     MOV  PCON,#IDL
                     EXITM
               ENDIF
                  JMP  $
            ENDM
```

7.3 Special Macro Operators

There are four special macro operators that are defined below:

% when the PERCENT sign prefaces a symbol in the parameter list, the symbol's value is passed to the macro's body instead of the symbol itself.

! when the EXCLAMATION POINT precedes a character, that character is handled as a literal and is passed to the macro body with the EXCLAMATION POINT removed. This is useful when it is necessary to pass a delimiter to the macro body. For example, in the following parameter list, the second parameter passed to the macro body would be a COMMA (,):

```
GENERATE_INST       75,!,,STK_VALUE
```

& when the AMPERSAND is used in the macro body, the symbols on both sides of it are concatenated together and the AMPERSAND is removed.

;; when double SEMI-COLONS are used in a macro definition, the comment preceded by the double SEMI_COLONS will not be saved and thus will not appear in the listing whenever the macro is invoked. Using the double SEMI-COLONS lowers the memory requirement in storing the macro definitions and should be used whenever possible.

Examples of using the above special macro operators follow in the "Using Macros" section.

7.4 Using Macros

This section section discusses several situations that arise using macros and how to handle them. In general the discussion uses examples to get the point across. First the macro definition is listed, then the source line program that will invoke the macro and finally how the macro was expanded by the Cross Assembler.

7.4.1 NESTING MACROS

The following shows a macro nested to a depth of three.
Remember, definitions cannot be nested. Macros must be defined before they are used in other macro definitions.

```
;MACRO DEFINITIONS

      GET_EXT_BYTE      MACRO EXT_ADDR
                        MOV   R0,#EXT_ADDR
                        MOVX  A,@R0
                        ENDM

      ADD_EXT_BYTES     MACRO EXT_DEST,EXT_SRC
                        GET   EXT_BYTE    EXT_DEST
                        MOV   R1,A
                        GET   EXT_BYTE    EXT_SRC
                        ADD   A,R1
                        ENDM
```

```
      ADD_DIRECT_BYTES  MACRO DESTINATION,SOURCE
                        IF (SMALL_MODEL)
                              MOV   A,SOURCE
                              ADD   A,DESTINATION
                              MOV   DESTINATION
                        ELSE
                              ADD_EXT_BYTES     DESTINATION,SOURCE
                              ENDIF
                        ENDM


;USAGE IN PROGRAM

      ADD_DIRECT_BYTES  127,128

;TRANSLATED MACRO


30 +1 ADD_DIRECT_BYTES  127,128
                  31 +1      IF (SMALL_MODEL)
                  32 +1           MOV    A,128
                  33 +1           ADD    A,127
                  34 +1           MOV    127
                  35 +1      ELSE
                  36 +2          ADD_EXT_BYTES  127,128
                  37 +3          GET_EXT_BYTE   127
0100 787F         38 +3          MOV    R0,#127
0102 E2           39 +3          MOVX   A,@R0
0103 F9           40 +2          MOV    R1,A
                  41 +3          GET_EXT_BYTE   128
0104 7880         42 +3          MOV    R0,#128
0106 E2           43 +3          MOVX   A,@R0
0107 29           44 +2          ADD    A,R1
0108 F2           45 +1          MOVX   @R0,A
                  46 +1      ENDIF
                  48
```

Two things should be pointed out from the above example. First, the order of the parameter list is important. You must maintain the order of parameters from the macro definition if the Cross Assembler is to translate the macro correctly.

Secondly, in order to pass parameters to nested macros, simply use the same parameter symbol in the parameter list of the definition. For example, the parameter DESTINATION was passed properly to the nested macros ADD_EXT_BYTES and GET_EXT_BYTE. This occurred because in the macro definition of ADD_DIRECT_BYTES, the parameter DESTINATION was specified in the parameter lists of both ADD_EXT_BYTES and GET_EXT_BYTE.

7.4.2 LABELS IN MACROS

You have two choices for specifying labels in a macro body. A label can either be passed to the body as a parameter or it can be generated within the body. The following example shows both ways.

```
;MACRO DEFINITION

MULTIPLE_SHIFT    MACRO LABEL,LABEL_SUFFIX,COUNTER,N
                  COUNTER SET  COUNTER+1    ;INCREMENT SUFFIX FOR NEXT USAGE
```

```
                    LABEL:      MOV   R0,#N
                    SHIFT&LABEL_SUFFIX:     RL    A
                    DJNZ  R0,SHIFT&LABEL_SUFFIX
                    ENDM


;USAGE IN PROGRAM

MULTIPLE_SHIFT      LOOP_SHIFT,%COUNT,COUNT,4

;TRANSLATED MACRO

                    15 +1 MULTIPLE_SHIFT LOOP_SHIFT,%COUNT,COUNT,4
0006                16 +1      COUNT   SET    COUNT+1
                    17 +1
0100 7804           18 +1 LOOP_SHIFT:  MOV    R0,#4
0102 23             19 +1 SHIFT5:      RL     A
0103 D8FD           20 +1              DJNZ    R0,SHIFT5
                    22
```

Points to note in the above example: 1) the double semi-colon caused the comment not to be listed in the translated macro; 2) the percent sign caused the value of COUNT (in this case the value 5) to be passed to the macro body instead of the symbol; and 3) the ampersand allowed two symbols to be concatenated to form the label SHIFT5.

CHAPTER 8

8 8051 CROSS ASSEMBLER ERROR CODES

8.1 Introduction

When the Cross Assembler encounters an error in the source program, it will emit an error message in the listing file. If the $NOPRINT control has been invoked, the error message will be output to the screen.

There are basically two types of errors that are encountered by the Cross Assembler, translation errors and I/O errors. I/O errors are usually fatal errors. However, whenever an error is detected, the Cross Assembler makes every effort possible to continue with the assembly.

If it is possible to recover from the error and continue assembling, the Cross Assembler will report the error, use a default condition and continue on its way. However, when a fatal error is encountered, it is impossible for the Cross Assembler to proceed. In this case, the Cross Assembler reports the error and then aborts the assembly process.

Fatal I/O error messages are displayed on the screen and are of the form:

FATAL ERROR opening <filename>

where <filename> would be replaced with the file designator initially entered or read from the source program. The cause of this error is usually obvious, typically a typographical error or the wrong drive specification.

Another fatal I/O error message is:

FATAL ERROR writing to <type> file

where <type> would be replaced with either "listing" or "object". The cause of this error is usually either a write protected disk or a full disk.

Translation error reports contain at least three lines. The first line is the source line in which the error was detected, the second line is a pointer to the character, symbol, expression or line that caused the error. The final line is the error message itself. There may be more than one error message, depending on the number of errors in the source line. An example of a source line with two errors in it follows:

```
0100 2323         26    START:      MOV   AB,@35
****-----------------------------------------^---^
****ERROR #20: Illegal operand
****ERROR #20: Illegal operand
```

The errors are pointed out by the up-arrows (^). For every up-arrow there will be an error message. Errors are ordered left to right, so the first error message corresponds to the left-most up-arrow and so on. The error message includes an error number and an description of the error. The error number can be used as an index to the more detailed error explanations that follow in this chapter.

After the Cross Assembler has completed its translation process, it will print an assembly complete message:

ASSEMBLY COMPLETE, nn ERRORS FOUND

If it was an error free assembly, in place of the "nn" above the word "NO" will be output. However, if errors were encountered during the assembly process, the "nn" will be replaced with the number of errors that were found (up to a maximum of 50). In this case, an error summary will follow in the listing file with all the errors that were reported during the assembly. An error summary looks like the following:

```
ERROR SUMMARY:
Line #26, ERROR #20: Illegal operand
Line #26, ERROR #20: Illegal operand
```

The same error message that occurred after the source line appears again prefaced by the source line number to aid in tracking down the error in the source listing.

8.2 Explanation of Error Messages

ERROR #1: Illegal character
This error occurs when the Cross Assembler encounters a character that is not part of its legal character set. The Cross Assembler character set can be found in Appendix D.

ERROR #2: Undefined symbol
This error occurs when the Cross Assembler tries to use a symbol that hasn't been defined. The two most common reasons for this error are typographical errors and forward references.

ERROR #3: Duplicate symbol
This error occurs when a previously defined symbol or a reserved symbol is attempted to be defined again. Refer to Appendix C for the reserved words. Also inspect the symbol in the symbol table listing. If the symbol doesn't appear there, you are using a reserved word. If the symbol does appear, its original definition will be listed.

ERROR #4: Illegal digit for radix A digit was encountered that is not part of the legal digits for the radix specified. Chapter 2 lists the legal digits for each radix available. Often this error occurs because a symbol was started with a number instead of a letter, question mark, or underscore.

ERROR #5: Number too large
The number specified, or the returned value of the expression, exceeds 16-bit precision. The largest value allowed is 65,535.

ERROR #6: Missing END directive
The source program must end with one and only one END directive. The END is placed after all the assembly line statements.

ERROR #7: Illegal opcode/directive after label
The symbol after a label is not an opcode nor a directive that allows labels. The only thing permitted on a line after a label is an instruction, the DS, DB or DW directives, or a comment. If none of these are found, this error will be reported.

ERROR #8: Illegal assembly line
The assembly line doesn't begin with a symbol, label, instruction mnemonic, control, directive, comment or null line. No attempt is made to translate such a line.

ERROR #9: Text beyond END directive

The END directive must be the last line of the source program. Any text beyond the END line will cause this error. Any such text is ignore. Text here is defined as any printable ASCII characters.

ERROR #10: Illegal or missing expression
A number, symbol or arithmetic expression was expected, but it was either found to be missing or the Cross Assembler was unable to evaluate it properly.

ERROR #11: Illegal or missing expression operator
An arithmetic operator was expected but it is either missing or it is not one of the legal operators specified in Chapter 2.

ERROR #12: Unbalanced parentheses
In evaluating an expression, the parentheses in the expression were found not to balance.

ERROR #13: Illegal or missing expression value
In evaluating an expression, the Cross Assembler expected to find either a number or a symbol, but it was either missing or illegal.

ERROR #14: Illegal literal expression
This error occurs when a null ASCII literal string is found. A null ASCII literal is nothing more than two apostrophes together ('') and is illegal.

ERROR #15: Expression stack overflow
The expression stack has a depth of 32 values. The expression being evaluated exceeds this depth. This is a very rare error. However, if you ever get it, divide the expression into two or more expressions using the EQU directive.

ERROR #16: Division by zero
The expression being evaluated includes an attempt to divide by zero.

ERROR #17: Illegal bit designator
A bit designator address was specified in the source program and it points to an illegal bit address. A bit designator contains a byte address, followed by a PERIOD, followed by the bit index into the byte address (e.g., ACC.7) as discussed in Chapter 2. This error can occur for one of two reasons. First, if the number or a symbol that is used to specify the byte address part of the bit designator is not a legal bit addressable address, ERROR #17 will occur. Second, if the bit index into the byte address exceeds the number 7, again ERROR #17 will be output.

ERROR #18: Target address exceeds relative address range
A Program Counter relative jump instruction (e.g., SJMP, JZ, JNC, etc.) was decoded with the target address of the jump exceeding the maximum possible forward jump of 127 bytes or the maximum possible backward jump of 128 bytes.

ERROR #20: Illegal operand
The operand specified is not a legal operand for the instruction. Review the legal operands allowed for the instruction.

ERROR #21: Illegal indirect register
R0 and R1 are the only primary legal indirect register. This error occurs when the indirect addressing mode designator (@) is not followed by either R0, R1 or symbols that were defined to be equivalent to either R0 or R1. This error can also occur in the MOVC A,@A+DPTR, MOVC A,@A+PC, MOVX A,@DPTR, MOVX @DPTR,A and the JMP @A+DPTR instructions if the operands after the indirect addressing mode designator (@) aren't specified properly.

ERROR #22: Missing operand delimiter
A COMMA operand delimiter is missing from the operand fields of the instruction.

ERROR #23: Illegal or missing directive
This error occurs when the Cross Assembler cannot find a legal directive. The most common cause of this error is due to leaving the COLON off a label. As a result, the following opcode mnemonic is attempted to be decoded as a directive.

ERROR #24: Attempting to EQUate a previously SET symbol Once a symbol is defined using the SET directive, it cannot be later redefined using the EQU directive.

ERROR #25: Attempting to SET a previously EQUated symbol
Once a symbol is defined using the EQU directive, it cannot be redefined. If you want the symbol to be redefineable, use the SET directive.

ERROR #26: Illegal SET/EQU expression
The expression following the SET or EQU directive is illegal. This typically occurs when an attempt is made to define a symbol to be equivalent to an implicit register other than A, R0, R1, R2, R3, R4, R5, R6 or R7.

ERROR #27: Illegal expression with forward reference
This error occurs when an expression contains a symbol that hasn't been defined yet. Move the symbol definition earlier in the source file.

ERROR #28: Address exceeds segment range
The address specified exceeds 255 and you are in the DSEG, BSEG, or ISEG.

ERROR #29: Expecting an EOL or COMMENT
The Cross Assembler has completed processing a legal assembly language line and expected the line to be terminated with either a COMMENT or a carriage return/line feed pair.

ERROR #30: Illegal directive with current active segment
The specified directive is not legal in the active segment. This can happen by trying to use the DBIT directive in other than the BSEG, or using the DS directive in the BSEG.

ERROR #31: Only two character string allowed
This error occurs using the DW directive. The maximum ASCII literal allowed in a DW specification is a two character string.

ERROR #32: Byte definition exceeds 255
This error occurs using the DB directive. The value specified in the DB specification cannot fit into a byte.

ERROR #33: Premature end of string
An ASCII literal string was not terminated properly with an apostrophe.

ERROR #34: Illegal register bank number
This error occurs when the number specified with the USING directive exceed 3. Legal register bank numbers are: 0, 1, 2, 3.

ERROR #35: Include file nesting exceeds 8
The maximum number of nested include files is eight. You will get this error if you exceed this limit.

ERROR #36: Illegal or missing argument
This error occurs when the syntax of a Cross Assembler control requires an argument and it was either incorrectly specified or is missing all together.

ERROR #37: Illegal control statement

The Cross Assembler does not recognize the specified control. The legal controls are detailed in Chapter 6.

ERROR #38: Unable to open file
The Cross Assembler is unable to open the file as specified. This is a fatal error which will abort the assembly process.

ERROR #39: Illegal file specification
The file specification is not a legal file designator. Refer to your DOS manual for a description of legal file designators. This is a fatal error which will abort the assembly process.

ERROR #40: Program synchronization error
This error occurs when the Cross Assembler is generating the object hex file and finds that the code segment location counter is not advancing properly. There are two cases where this can happen. First, if the source program uses ORG directives and they are not placed in ascending order. Second, if a generic CALL or JMP is made to a forward reference that is actually defined later in the program to be a backward reference. For example, the following code sequence will cause this error due to the second reason:

```
BACK_REF:   NOP
            CALL  FORWARD_REF
FORWARD_REF EQU   BACK_REF
```

During the first pass, the generic CALL will be replaced with a 3-byte LCALL instruction. During the second pass, the generic CALL will be replaced with a 2-byte ACALL instruction. To prevent this kind of problem, use the generic CALLs and JMPs with labeled targets, not EQU or SET defined symbols.

ERROR #41: Insufficient memory
This error occurs when there isn't enough memory to hold all the symbols that have been generated by the source program. If you have 96 Kbytes or more of RAM this will be a very rare error. Only a massive source program or numerous large macros could potentially cause this error. However, if this error does occur, your best bet is to either buy more memory or to break up your program into smaller pieces and share common symbols with a common $INCLUDE file.

ERROR #42: More errors detected, not listed
The internal error buffer can hold 50 errors. If more than 50 errors occur, only the first 50 will be reported.

ERROR #43: ENDIF without IF
The terminator of a conditional assembly block (ENDIF) was recognized without seeing a matching IF.

ERROR #44: Missing ENDIF
A conditional assembly block was begun with an IF statement, but no matching ENDIF was detected.

ERROR #45: Illegal or missing macro name
The MACRO keyword was recognized, but the symbol that is supposed to precede the MACRO keyword was missing, an illegal symbol or a duplicate symbol.

ERROR #46: Macro nesting too deep
Macros can be nested to a depth of 9 levels. Exceeding this limit will cause this error.

ERROR #47: Number of parameters doesn't match definition

In attempting to use a macro, the number of parameters in the parameter list does not equal the number of parameters specified in the macro definition. They must match.

ERROR #48: Illegal parameter specification
This error typically occurs when a previously defined symbol is used in the parameter list of the macro definition.

ERROR #49: Too many parameters
The maximum number of parameters in a macro parameter list is sixteen. This error occurs when you exceed that limit.

ERROR #50: Line exceeds 255 characters
The maximum length of a source line is 255 characters. If a carriage return/line feed pair is not detected in the first 256 characters of a line, this error is reported and the line is truncated at 255 characters.

APPENDICES

9 APPENDICES

APPENDIX A - SAMPLE PROGRAM AND LISTING

A.1 Source File

```
;
;  8-bit by 8-bit signed multiply--byte signed multiply
;
;    This routine takes the signed byte in multiplicand and
;    multiplies it by the signed byte in multiplier and places
;    the signed 16-bit product in product_high and product_low.
;
;    This routine assumes 2s complement representation of signed
;    numbers.  The maximum numbers possible are then -128 and
;    +127.  Multiplying the possible maximum numbers together
;    easily fits into a 16-bit product, so no overflow test is
;    done on the answer.
;
;    Registers altered by routine: A, B, PSW.
;
;
;  Primary controls
$MOD51
$TITLE(BYTE SIGNED MULTIPLY)
$DATE(JUL-30-84)
$PAGEWIDTH(132)
$OBJECT(B:BMULB.OBJ)
;
;
;  Variable declarations
;
sign_flag  BIT     0F0H             ;sign of product
multiplier DATA    030H             ;8-bit multiplier
multiplicand       DATA    031H    ;8-bit multiplicand
product_high       DATA    032H    ;high byte of 16-bit answer
product_low        DATA    033H    ;low byte of answer
;
;
;
ORG     100H                      ;arbitrary start
;
byte_signed_multiply:
            CLR     sign_flag       ;reset sign
            MOV     A,multiplier    ;put multiplier in accumulator
            JNB     ACC.7,positive  ;test sign bit of multiplier
            CPL     A               ;negative--complement and
            INC     A               ;add 1 to convert to positive
            SETB    sign_flag       ;and set sign flag
;
positive:   MOV     B,multiplicand  ;put multiplicand in B register
            JNB     B.7,multiply    ;test sign bit of multiplicand
            XRL     B,#0FFh         ;negative--complement and
            INC     B               ;add 1 to convert to positive
            CPL     sign_flag       ;complement sign flag
;
multiply:   MUL     AB              ;do unsigned multiplication
```

```
;
sign_test:  JNB     sign_flag,byte_signed_exit   ;if positive,done
            XRL     B,#0FFh         ;else have to complement both
            CPL     A               ;bytes of the product and inc
            ADD     A,#1            ;add here because inc doesn't
            JNC     byte_signed_exit      ;set the carry flag
            INC     B               ;if add overflowed A, inc the
;high byte
byte_signed_exit:
            MOV     product_high,B  ;save the answer
            MOV     product_low,A
;
            RET                     ;and return
            END
```

A.2 Source File Listing

```
BMULB          BYTE SIGNED MULTIPLY                                                     JUL-30-84   PAGE 1


                          1    ;
                          2    ;  8-bit by 8-bit signed multiply--byte signed multiply
                          3    ;
                          4    ;    This routine takes the signed byte in multiplicand and
                          5    ;    multiplies it by the signed byte in multiplier and places
                          6    ;    the signed 16-bit product in product_high and product_low.
                          7    ;
                          8    ;    This routine assumes 2s complement representation of signed
                          9    ;    numbers.  The maximum numbers possible is then -128 and +127.
                         10    ;    Multiplying the possible maximum numbers together easily fits
                         11    ;    in a 16-bit product, so no overflow test is done on the answer.
                         12    ;
                         13    ;    Registers altered by routine: A, B, PSW.
                         14    ;
                         15    ;
                         16    ;  Primary controls
                         17    $MOD51
                         18    $TITLE(BYTE SIGNED MULTIPLY)
                         19    $DATE(JUL-30-84)
                         20    $PAGEWIDTH(132)
                         21    $OBJECT(B:BMULB.OBJ)
                         22    ;
                         23    ;
                         24    ;  Variable declarations
                         25    ;
00F0                 26     sign_flag       BIT     0F0H            ;sign of product
0030                 27     multiplier      DATA    030H            ;8-bit multiplier
0031                 28     multiplicand    DATA    031H            ;8-bit multiplicand
0032                 29     product_high    DATA    032H            ;high byte of 16-bit answer
0033                 30     product_low     DATA    033H            ;low byte of answer
                         31    ;
                         32    ;
                         33    ;
0100                 34            ORG     100H                    ;arbitrary start
                         35    ;
0100                 36     byte_signed_multiply:
0100 C2F0            37                     CLR     sign_flag       ;reset sign
0102 E530            38                     MOV     A,multiplier    ;put multiplier in accumulator
0104 30E704          39                     JNB     ACC.7,positive  ;test sign bit of multiplier
0107 F4              40                     CPL     A               ;negative--complement and



BMULB          BYTE SIGNED MULTIPLY                                                     JUL-30-84   PAGE 2

0108 04              41                     INC     A               ;add 1 to convert to positive
0109 D2F0            42                     SETB    sign_flag       ;and set sign flag
                         43    ;
010B 8531F0          44     positive:       MOV     B,multiplicand  ;put multiplicand in B register
010E 30F707          45                     JNB     B.7,multiply    ;test sign bit of multiplicand
0111 63F0FF          46                     XRL     B,#0FFh         ;negative--complement and
0114 05F0            47                     INC     B               ;add 1 to convert to positive
0116 B2F0            48                     CPL     sign_flag       ;complement sign flag
                         49    ;
0118 A4              50     multiply:       MUL     AB              ;do unsigned multiplication
                         51    ;
0119 30F00A          52     sign_test:      JNB     sign_flag,byte_signed_exit      ;if positive,done
011C 63F0FF          53                     XRL     B,#0FFh         ;else have to complement both
011F F4              54                     CPL     A               ;bytes of the product and inc
0120 2401            55                     ADD     A,#1            ;need add here because inc doesn't set
0122 5002            56                     JNC     byte_signed_exit        ;the carry flag
0124 05F0            57                     INC     B               ;if add overflowed A, inc the high byte
                         58    ;
0126                 59     byte_signed_exit:
0126 85F032          60                     MOV     product_high,B  ;save the answer
0129 F533            61                     MOV     product_low,A
                         62    ;
012B 22              63                     RET                     ;and return
                         64             END


ASSEMBLY COMPLETE, 0 ERRORS FOUND
BMULB          BYTE SIGNED MULTIPLY                                                     JUL-30-84   PAGE 3

ACC                   D ADDR  00E0H  PREDEFINED
B                     D ADDR  00F0H  PREDEFINED
BYTE_SIGNED_EXIT      C ADDR  0126H
BYTE_SIGNED_MULTIPLY  C ADDR  0100H  NOT USED
MULTIPLICAND          D ADDR  0031H
MULTIPLIER            D ADDR  0030H
MULTIPLY              C ADDR  0118H
POSITIVE              C ADDR  010BH
PRODUCT_HIGH          D ADDR  0032H
PRODUCT_LOW           D ADDR  0033H
SIGN_FLAG             B ADDR  00F0H
SIGN_TEST             C ADDR  0119H  NOT USED
```

APPENDIX B - PRE-DEFINED BYTE AND BIT ADDRESSES

The following tables detail the pre-defined byte and bit addresses for the 8051/8031 microcontrollers supported by the MetaLink family of emulators. Proliferation parts are delimited from the standard MCS-51 definitions by asterisk ("*") boxes.

This list covers these microcontrollers:

```
8044  8031   8032   8051   8052   8053  80C154  80C321
8344  80C31  80C32  8751   8752   8753  83C154  80C521
8744                80C51  80C52        85C154  87C521
                                                87C51

80C321  80C51FA(80C252)  80C452  80C152JA/JB/JC/JD    80C851
80C541  83C51FA(83C252)  83C452  83C152JA/JC          83C851
87C541  87C51FA(87C252)  87C452

80C451  80C652  80C552  83C751  83C752  80512  80515  80C515  80C517
83C451  83C652  83C552  87C751  87C752  80532  80535  80C535  80C537
87C451  87C652  87C552
```

B.1 Pre-defined Byte Addresses

```
P0        DATA      080H      ;PORT 0
SP        DATA      081H      ;STACK POINTER
DPL       DATA      082H      ;DATA POINTER - LOW BYTE
DPH       DATA      083H      ;DATA POINTER - HIGH BYTE

************************************************************************
for the 80C321/80C521
DPL1      DATA      084H      ;DATA POINTER LOW 1
DPH1      DATA      085H      ;DATA POINTER HIGH 1
DPS       DATA      086H      ;DATA POINTER SELECTION
************************************************************************
************************************************************************
for the 83C152/80C152
GMOD      DATA      084H      ;GSC MODE
TFIFO     DATA      085H      ;GSC TRANSMIT BUFFER
************************************************************************
************************************************************************
for the 80C517/80C537
WDTREL    DATA      086H      ;WATCHDOG TIMER RELOAD REG
************************************************************************

PCON      DATA      087H      ;POWER CONTROL
TCON      DATA      088H      ;TIMER CONTROL
TMOD      DATA      089H      ;TIMER MODE
TL0       DATA      08AH      ;TIMER 0 - LOW BYTE
TL1       DATA      08BH      ;TIMER 1 - LOW BYTE

************************************************************************
for the 83C751/83C752
RTL       DATA      08BH      ;TIMER 0 - LOW BYTE RELOAD
************************************************************************

TH0       DATA      08CH      ;TIMER 0 - HIGH BYTE
TH1       DATA      08DH      ;TIMER 1 - HIGH BYTE
```

```
***********************************************************************
for the 83C751/83C752
RTH        DATA      08DH      ;TIMER 0 - HIGH BYTE RELOAD
***********************************************************************
***********************************************************************
for the 83C752
PWM        DATA      08EH      ;PULSE WIDTH MODULATION
***********************************************************************

P1         DATA      090H      ;PORT 1

***********************************************************************
for the 83C152/80C152
P5         DATA      091H      ;PORT 5
DCON0      DATA      092H      ;DMA CONTROL 0
DCON1      DATA      093H      ;DMA CONTROL 1
BAUD       DATA      094H      ;GSC BAUD RATE
ADR0       DATA      095H      ;GSC MATCH ADDRESS 0
***********************************************************************
***********************************************************************
for the 80C452/83C452
DCON0      DATA      092H      ;DMA CONTROL 0
DCON1      DATA      093H      ;DMA CONTROL 1
***********************************************************************
***********************************************************************
for the 80C517/80C537
DPSEL      DATA      092H      ;DATA POINTER SELECT REGISTER
***********************************************************************

SCON       DATA      098H      ;SERIAL PORT CONTROL
SBUF       DATA      099H      ;SERIAL PORT BUFFER

***********************************************************************
for the 83C751/83C752
I2CON      DATA      098H      ;I2C CONTROL
I2DAT      DATA      099H      ;I2C DATA
***********************************************************************
***********************************************************************
for the 80C517/80C537
IEN2       DATA      09AH      ;INTERRUPT ENABLE REGISTER 2
S1CON      DATA      09BH      ;SERIAL PORT CONTROL 1
S1BUF      DATA      09CH      ;SERIAL PORT BUFFER 1
S1REL      DATA      09DH      ;SERIAL RELOAD REG 1
***********************************************************************

P2         DATA      0A0H      ;PORT 2
IE         DATA      0A8H      ;INTERRUPT ENABLE

***********************************************************************
for the 80C51FA/83C51FA(83C252/80C252)
SADDR      DATA      0A9H      ;SLAVE INDIVIDUAL ADDRESS
***********************************************************************
***********************************************************************
for the 80515/80535 and 80C517/80C537
IP0        DATA      0A9H      ;INTERRUPT PRIORITY REGISTER 0
***********************************************************************
***********************************************************************
for the 80C321/80C521
WDS        DATA      0A9H      ;WATCHDOG SELECTION
WDK        DATA      0AAH      ;WATCHDOG KEY
```

```
***********************************************************************
***********************************************************************
for the 83C152/80C152
P6         DATA      0A1H      ;PORT 6
SARL0      DATA      0A2H      ;DMA SOURCE ADDR. 0 (LOW)
SARH0      DATA      0A3H      ;DMA SOURCE ADDR. 0 (HIGH)
IFS        DATA      0A4H      ;GSC INTERFRAME SPACING
ADR1       DATA      0A5H      ;GSC MATCH ADDRESS 1
***********************************************************************
***********************************************************************
for the 80C452/83C452
SARL0      DATA      0A2H      ;DMA SOURCE ADDR. 0 (LOW)
SARH0      DATA      0A3H      ;DMA SOURCE ADDR. 0 (HIGH)
***********************************************************************
***********************************************************************
for the 80C552/83C552
CML0       DATA      0A9H      ;COMPARE 0 - LOW BYTE
CML1       DATA      0AAH      ;COMPARE 1 - LOW BYTE
CML2       DATA      0ABH      ;COMPARE 2 - LOW BYTE
CTL0       DATA      0ACH      ;CAPTURE 0 - LOW BYTE
CTL1       DATA      0ADH      ;CAPTURE 1 - LOW BYTE
CTL2       DATA      0AEH      ;CAPTURE 2 - LOW BYTE
CTL3       DATA      0AFH      ;CAPTURE 3 - LOW BYTE
***********************************************************************

P3         DATA      0B0H      ;PORT 3

***********************************************************************
for the 83C152/80C152
SARL1      DATA      0B2H      ;DMA SOURCE ADDR. 1 (LOW)
SARH1      DATA      0B3H      ;DMA SOURCE ADDR. 1 (HIGH)
SLOTTM     DATA      0B4H      ;GSC SLOT TIME
ADR2       DATA      0B5H      ;GSC MATCH ADDRESS 2
***********************************************************************
***********************************************************************
for the 80C452/83C452
SARL1      DATA      0B2H      ;DMA SOURCE ADDR. 1 (LOW)
SARH1      DATA      0B3H      ;DMA SOURCE ADDR. 1 (HIGH)
***********************************************************************

IP         DATA      0B8H      ;INTERRUPT PRIORITY

***********************************************************************
for the 80C51FA/83C51FA(83C252/80C252)
SADEN      DATA      0B9H      ;SLAVE ADDRESS ENABLE
***********************************************************************
***********************************************************************
for the 80515/80535 and 80C517/80C537
IP1        DATA      0B9H      ;INTERRUPT PRIORITY REGISTER 1
IRCON      DATA      0C0H      ;INTERRUPT REQUEST CONTROL
CCEN       DATA      0C1H      ;COMPARE/CAPTURE ENABLE
CCL1       DATA      0C2H      ;COMPARE/CAPTURE REGISTER 1  - LOW BYTE
CCH1       DATA      0C3H      ;COMPARE/CAPTURE REGISTER 1  - HIGH BYTE
CCL2       DATA      0C4H      ;COMPARE/CAPTURE REGISTER 2  - LOW BYTE
CCH2       DATA      0C5H      ;COMPARE/CAPTURE REGISTER 2  - HIGH BYTE
CCL3       DATA      0C6H      ;COMPARE/CAPTURE REGISTER 3  - LOW BYTE
CCH3       DATA      0C7H      ;COMPARE/CAPTURE REGISTER 3  - HIGH BYTE
T2CON      DATA      0C8H      ;TIMER 2 CONTROL
CRCL       DATA      0CAH      ;COMPARE/RELOAD/CAPTURE - LOW BYTE
CRCH       DATA      0CBH      ;COMPARE/RELOAD/CAPTURE - HIGH BYTE
```

```
TL2       DATA      0CCH      ;TIMER 2 - LOW BYTE
TH2       DATA      0CDH      ;TIMER 2 - HIGH BYTE
***********************************************************************
***********************************************************************
for the 80C517/80C537
CC4EN     DATA      0C9H      ;COMPARE/CAPTURE 4 ENABLE
CCL4      DATA      0CEH      ;COMPARE/CAPTURE REGISTER 4  - LOW BYTE
CCH4      DATA      0CFH      ;COMPARE/CAPTURE REGISTER 4  - HIGH BYTE
***********************************************************************
***********************************************************************
for the RUPI-44
STS       DATA      0C8H      ;SIU STATUS REGISTER
SMD       DATA      0C9H      ;SERIAL MODE
RCB       DATA      0CAH      ;RECEIVE CONTROL BYTE
RBL       DATA      0CBH      ;RECEIVE BUFFER LENGTH
RBS       DATA      0CCH      ;RECEIVE BUFFER START
RFL       DATA      0CDH      ;RECEIVE FIELD LENGTH
STAD      DATA      0CEH      ;STATION ADDRESS
DMA_CNT   DATA      0CFH      ;DMA COUNT
***********************************************************************
***********************************************************************
for the 8052/8032, 80C51FA/83C51FA(83C252/80C252), 80C154/83C154
T2CON     DATA      0C8H      ;TIMER 2 CONTROL
***********************************************************************
***********************************************************************
for the 80C51FA/83C51FA(83C252/80C252)
T2MOD     DATA      0C9H      ;TIMER 2 MODE CONTROL
***********************************************************************
***********************************************************************
for the 8052/8032, 80C51FA/83C51FA(83C252/80C252), 80C154/83C154
RCAP2L    DATA      0CAH      ;TIMER 2 CAPTURE REGISTER, LOW BYTE
RCAP2H    DATA      0CBH      ;TIMER 2 CAPTURE REGISTER, HIGH BYTE
TL2       DATA      0CCH      ;TIMER 2 - LOW BYTE
TH2       DATA      0CDH      ;TIMER 2 - HIGH BYTE
***********************************************************************
***********************************************************************
for the 83C152/80C152
P4        DATA      0C0H      ;PORT 4
DARL0     DATA      0C2H      ;DMA DESTINATION ADDR. 0 (LOW)
DARH0     DATA      0C3H      ;DMA DESTINATION ADDR. 0 (HIGH)
BKOFF     DATA      0C4H      ;GSC BACKOFF TIMER
ADR3      DATA      0C5H      ;GSC MATCH ADDRESS 3
IEN1      DATA      0C8H      ;INTERRUPT ENABLE REGISTER 1
***********************************************************************
***********************************************************************
for the 80C452/83C452
P4        DATA      0C0H      ;PORT 4
DARL0     DATA      0C2H      ;DMA DESTINATION ADDR. 0 (LOW)
DARH0     DATA      0C3H      ;DMA DESTINATION ADDR. 0 (HIGH)
***********************************************************************
***********************************************************************
for the 80C451/83C451
P4        DATA      0C0H      ;PORT 4
P5        DATA      0C8H      ;PORT 5
***********************************************************************
***********************************************************************
for the 80512/80532
IRCON     DATA      0C0H      ;INTERRUPT REQUEST CONTROL
***********************************************************************
```

```
*************************************************************************
for the 80C552/83C552
P4          DATA      0C0H      ;PORT 4
P5          DATA      0C4H      ;PORT 5
ADCON       DATA      0C5H      ;A/D CONVERTER CONTROL
ADCH        DATA      0C6H      ;A/D CONVERTER HIGH BYTE
TM2IR       DATA      0C8H      ;T2 INTERRUPT FLAGS
CMH0        DATA      0C9H      ;COMPARE 0 - HIGH BYTE
CMH1        DATA      0CAH      ;COMPARE 1 - HIGH BYTE
CMH2        DATA      0CBH      ;COMPARE 2 - HIGH BYTE
CTH0        DATA      0CCH      ;CAPTURE 0 - HIGH BYTE
CTH1        DATA      0CDH      ;CAPTURE 1 - HIGH BYTE
CTH2        DATA      0CEH      ;CAPTURE 2 - HIGH BYTE
CTH3        DATA      0CFH      ;CAPTURE 3 - HIGH BYTE
*************************************************************************

PSW         DATA      0D0H      ;PROGRAM STATUS WORD

*************************************************************************
for the RUPI-44
NSNR        DATA      0D8H      ;SEND COUNT/RECEIVE COUNT
SIUST       DATA      0D9H      ;SIU STATE COUNTER
TCB         DATA      0DAH      ;TRANSMIT CONTROL BYTE
TBL         DATA      0DBH      ;TRANSMIT BUFFER LENGTH
TBS         DATA      0DCH      ;TRANSMIT BUFFER START
FIFO0       DATA      0DDH      ;THREE BYTE FIFO
FIFO1       DATA      0DEH
FIFO2       DATA      0DFH
*************************************************************************
*************************************************************************
for the 80C51FA/83C51FA(83C252/80C252)
CCON        DATA      0D8H      ;CONTROL COUNTER
CMOD        DATA      0D9H      ;COUNTER MODE
CCAPM0      DATA      0DAH      ;COMPARE/CAPTURE MODE FOR PCA MODULE 0
CCAPM1      DATA      0DBH      ;COMPARE/CAPTURE MODE FOR PCA MODULE 1
CCAPM2      DATA      0DCH      ;COMPARE/CAPTURE MODE FOR PCA MODULE 2
CCAPM3      DATA      0DDH      ;COMPARE/CAPTURE MODE FOR PCA MODULE 3
CCAPM4      DATA      0DEH      ;COMPARE/CAPTURE MODE FOR PCA MODULE 4
*************************************************************************
*************************************************************************
for the 80515/80535
ADCON       DATA      0D8H      ;A/D CONVERTER CONTROL
ADDAT       DATA      0D9H      ;A/D CONVERTER DATA
DAPR        DATA      0DAH      ;D/A CONVERTER PROGRAM REGISTER
*************************************************************************
*************************************************************************
for the 83C152/80C152
DARL1       DATA      0D2H      ;DMA DESTINATION ADDR. 1 (LOW)
DARH1       DATA      0D3H      ;DMA DESTINATION ADDR. 1 (HIGH)
TCDCNT      DATA      0D4H      ;GSC TRANSMIT COLLISION COUNTER
AMSK0       DATA      0D5H      ;GSC ADDRESS MASK 0
TSTAT       DATA      0D8H      ;TRANSMIT STATUS (DMA & GSC)
*************************************************************************
*************************************************************************
for the 80C452/83C452
DARL1       DATA      0D2H      ;DMA DESTINATION ADDR. 1 (LOW)
DARH1       DATA      0D3H      ;DMA DESTINATION ADDR. 1 (HIGH)
*************************************************************************

*************************************************************************
```

```
for the 80C451/83C451
P6        DATA      0D8H      ;PORT 6
************************************************************************
************************************************************************
for the 80512/80532
ADCON     DATA      0D8H      ;A/D CONVERTER CONTROL
ADDAT     DATA      0D9H      ;A/D CONVERTER DATA
DAPR      DATA      0DAH      ;D/A CONVERTER PROGRAM REGISTER
P6        DATA      0DBH      ;PORT 6
************************************************************************
************************************************************************
for the 83C751/
I2CFG     DATA      0D8H      ;I2C CONFIGURATION
************************************************************************
************************************************************************
for the 80C552/83C552 and 80C652/83C652
S1CON     DATA      0D8H      ;SERIAL 1 CONTROL
S1STA     DATA      0D9H      ;SERIAL 1 STATUS
S1DAT     DATA      0DAH      ;SERIAL 1 DATA
S1ADR     DATA      0DBH      ;SERIAL 1 SLAVE ADDRESS
************************************************************************
************************************************************************
for the 80C517/80C537
CML0      DATA      0D2H      ;COMPARE REGISTER 0  - LOW BYTE
CMH0      DATA      0D3H      ;COMPARE REGISTER 0  - HIGH BYTE
CML1      DATA      0D4H      ;COMPARE REGISTER 1  - LOW BYTE
CMH1      DATA      0D5H      ;COMPARE REGISTER 1  - HIGH BYTE
CML2      DATA      0D6H      ;COMPARE REGISTER 2  - LOW BYTE
CMH2      DATA      0D7H      ;COMPARE REGISTER 2  - HIGH BYTE
ADCON0    DATA      0D8H      ;A/D CONVERTER CONTROL 0
ADDAT     DATA      0D9H      ;A/D CONVERTER DATA
DAPR      DATA      0DAH      ;D/A CONVERTER PROGRAM REGISTER
P7        DATA      0DBH      ;PORT 7
ADCON1    DATA      0DCH      ;A/D CONVERTER CONTROL 1
P8        DATA      0DDH      ;PORT 8
CTRELL    DATA      0DEH      ;COM TIMER REL REG - LOW BYTE
CTRELH    DATA      0DFH      ;COM TIMER REL REG - HIGH BYTE
************************************************************************

ACC       DATA      0E0H      ;ACCUMULATOR

************************************************************************
for the 83C152/80C152
BCRL0     DATA      0E2H      ;DMA BYTE COUNT 0 (LOW)
BCRH0     DATA      0E3H      ;DMA BYTE COUNT 0 (HIGH)
PRBS      DATA      0E4H      ;GSC PSEUDO-RANDOM SEQUENCE
AMSK1     DATA      0E5H      ;GSC ADDRESS MASK 1
RSTAT     DATA      0E8H      ;RECEIVE STATUS (DMA & GSC)
************************************************************************
************************************************************************
for the 80C452/83C452
BCRL0     DATA      0E2H      ;DMA BYTE COUNT 0 (LOW)
BCRH0     DATA      0E3H      ;DMA BYTE COUNT 0 (HIGH)
HSTAT     DATA      0E6H      ;HOST STATUS
HCON      DATA      0E7H      ;HOST CONTROL
SLCON     DATA      0E8H      ;SLAVE CONTROL
SSTAT     DATA      0E9H      ;SLAVE STATUS
IWPR      DATA      0EAH      ;INPUT WRITE POINTER
IRPR      DATA      0EBH      ;INPUT READ POINTER
CBP       DATA      0ECH      ;CHANNEL BOUNDARY POINTER
```

```
FIN       DATA    0EEH     ;FIFO IN
CIN       DATA    0EFH     ;COMMAND IN
***********************************************************************
***********************************************************************
for the 80515/80535
P4        DATA    0E8H     ;PORT 4
***********************************************************************
***********************************************************************
for the 80C451/83C451
CSR       DATA    0E8H     ;CONTROL STATUS
***********************************************************************
***********************************************************************
for the 80512/80532
P4        DATA    0E8H     ;PORT 4
***********************************************************************
***********************************************************************
for the 80C552/83C552
IEN1      DATA    0E8H     ;INTERRUPT ENABLE REGISTER 1
TM2CON    DATA    0EAH     ;T2 COUNTER CONTROL
CTCON     DATA    0EBH     ;CAPTURE CONTROL
TML2      DATA    0ECH     ;TIMER 2 - LOW BYTE
TMH2      DATA    0EDH     ;TIMER 2 - HIGH BYTE
STE       DATA    0EEH     ;SET ENABLE
RTE       DATA    0EFH     ;RESET/TOGGLE ENABLE
***********************************************************************
***********************************************************************
for the 80C51FA/83C51FA(83C252/80C252)
CL        DATA    0E9H     ;CAPTURE BYTE LOW
CCAP0L    DATA    0EAH     ;COMPARE/CAPTURE 0 LOW BYTE
CCAP1L    DATA    0EBH     ;COMPARE/CAPTURE 1 LOW BYTE
CCAP2L    DATA    0ECH     ;COMPARE/CAPTURE 2 LOW BYTE
CCAP3L    DATA    0EDH     ;COMPARE/CAPTURE 3 LOW BYTE
CCAP4L    DATA    0EEH     ;COMPARE/CAPTURE 4 LOW BYTE
***********************************************************************
***********************************************************************
for the 80C517/80C537
CTCON     DATA    0E1H     ;COM TIMER CONTROL REG
CML3      DATA    0E2H     ;COMPARE REGISTER 3  - LOW BYTE
CMH3      DATA    0E3H     ;COMPARE REGISTER 3  - HIGH BYTE
CML4      DATA    0E4H     ;COMPARE REGISTER 4  - LOW BYTE
CMH4      DATA    0E5H     ;COMPARE REGISTER 4  - HIGH BYTE
CML5      DATA    0E6H     ;COMPARE REGISTER 5  - LOW BYTE
CMH5      DATA    0E7H     ;COMPARE REGISTER 5  - HIGH BYTE
P4        DATA    0E8H     ;PORT 4
MD0       DATA    0E9H     ;MUL/DIV REG 0
MD1       DATA    0EAH     ;MUL/DIV REG 1
MD2       DATA    0EBH     ;MUL/DIV REG 2
MD3       DATA    0ECH     ;MUL/DIV REG 3
MD4       DATA    0EDH     ;MUL/DIV REG 4
MD5       DATA    0EEH     ;MUL/DIV REG 5
ARCON     DATA    0EFH     ;ARITHMETIC CONTROL REG
***********************************************************************

B         DATA    0F0H     ;MULTIPLICATION REGISTER

***********************************************************************
for the 80C154/83C154
IOCON     DATA    0F8H     ;I/O CONTROL REGISTER
***********************************************************************
***********************************************************************
```

```
for the 83C152/80C152
BCRL1     DATA     0F2H     ;DMA BYTE COUNT 1 (LOW)
BCRH1     DATA     0F3H     ;DMA BYTE COUNT 1 (HIGH)
RFIFO     DATA     0F4H     ;GSC RECEIVE BUFFER
MYSLOT    DATA     0F5H     ;GSC SLOT ADDRESS
IPN1      DATA     0F8H     ;INTERRUPT PRIORITY REGISTER 1
***********************************************************************
***********************************************************************
for the 83C851/80C851
EADRL     DATA     0F2H     ;EEPROM Address Register - Low Byte
EADRH     DATA     0F3H     ;EEPROM Address Register - High Byte
EDAT      DATA     0F4H     ;EEPROM Data Register
ETIM      DATA     0F5H     ;EEPROM Timer Register
ECNTRL    DATA     0F6H     ;EEPROM Control Register
***********************************************************************
***********************************************************************
for the 80C452/83C452
BCRL1     DATA     0F2H     ;DMA BYTE COUNT 1 (LOW)
BCRH1     DATA     0F3H     ;DMA BYTE COUNT 1 (HIGH)
ITHR      DATA     0F6H     ;INPUT FIFO THRESHOLD
OTHR      DATA     0F7H     ;OUTPUT FIFO THRESHOLD
IEP       DATA     0F8H     ;INTERRUPT PRIORITY
MODE      DATA     0F9H     ;MODE
ORPR      DATA     0FAH     ;OUTPUT READ POINTER
OWPR      DATA     0FBH     ;OUTPUT WRITE POINTER
IMIN      DATA     0FCH     ;IMMEDIATE COMMAND IN
IMOUT     DATA     0FDH     ;IMMEDIATE COMMAND OUT
FOUT      DATA     0FEH     ;FIFO OUT
COUT      DATA     0FFH     ;COMMAND OUT
***********************************************************************
***********************************************************************
for the 80515/80535
P5        DATA     0F8H     ;PORT 5
***********************************************************************
***********************************************************************
for the 80512/80532
P5        DATA     0F8H     ;PORT 5
***********************************************************************
***********************************************************************
for the 83C751/83C752
I2STA     DATA     0F8H     ;I2C STATUS
***********************************************************************
***********************************************************************
for the 80C552/83C552
IP1       DATA     0F8H     ;INTERRUPT PRIORITY REGISTER 1
PWM0      DATA     0FCH     ;PULSE WIDTH REGISTER 0
PWM1      DATA     0FDH     ;PULSE WIDTH REGISTER 1
PWMP      DATA     0FEH     ;PRESCALER FREQUENCY CONTROL
T3        DATA     0FFH     ;T3 - WATCHDOG TIMER
***********************************************************************
***********************************************************************
for the 80C517/80C537
CMEN      DATA     0F6H     ;COMPARE ENABLE
CML6      DATA     0F2H     ;COMPARE REGISTER 6  - LOW BYTE
CMH6      DATA     0F3H     ;COMPARE REGISTER 6  - HIGH BYTE
CML7      DATA     0F4H     ;COMPARE REGISTER 7  - LOW BYTE
CMH7      DATA     0F5H     ;COMPARE REGISTER 7  - HIGH BYTE
CMSEL     DATA     0F7H     ;COMPARE INPUT REGISTER
P5        DATA     0F8H     ;PORT 5
P6        DATA     0FAH     ;PORT 6
```

```
***********************************************************************
***********************************************************************
for the 80C51FA/83C51FA(83C252/80C252)
CH          DATA      0F9H      ;CAPTURE HIGH BYTE
CCAP0H      DATA      0FAH      ;COMPARE/CAPTURE 0 HIGH BYTE
CCAP1H      DATA      0FBH      ;COMPARE/CAPTURE 1 HIGH BYTE
CCAP2H      DATA      0FCH      ;COMPARE/CAPTURE 2 HIGH BYTE
CCAP3H      DATA      0FDH      ;COMPARE/CAPTURE 3 HIGH BYTE
CCAP4H      DATA      0FEH      ;COMPARE/CAPTURE 4 HIGH BYTE
***********************************************************************
***********************************************************************
for the 83C752
PWENA       DATA      0FEH      ;PULSE WIDTH ENABLE
***********************************************************************
```

B.2 Pre-defined Bit Addresses

```
************************************************************************
for the 83C751/83C752
SCL       BIT       080H      ;P0.0 - I2C SERIAL CLOCK
SDA       BIT       081H      ;P0.1 - I2C SERIAL DATA
************************************************************************

IT0       BIT       088H      ;TCON.0 - EXT. INTERRUPT 0 TYPE
IE0       BIT       089H      ;TCON.1 - EXT. INTERRUPT 0 EDGE FLAG
IT1       BIT       08AH      ;TCON.2 - EXT. INTERRUPT 1 TYPE
IE1       BIT       08BH      ;TCON.3 - EXT. INTERRUPT 1 EDGE FLAG
TR0       BIT       08CH      ;TCON.4 - TIMER 0 ON/OFF CONTROL
TF0       BIT       08DH      ;TCON.5 - TIMER 0 OVERFLOW FLAG
TR1       BIT       08EH      ;TCON.6 - TIMER 1 ON/OFF CONTROL
TF1       BIT       08FH      ;TCON.7 - TIMER 1 OVERFLOW FLAG

************************************************************************
for the 83C751/83C752
C/T       BIT       08EH      ;TCON.6 - COUNTER OR TIMER OPERATION
GATE      BIT       08FH      ;TCON.7 - GATE TIMER
************************************************************************
************************************************************************
for the 80515/80535
INT3      BIT       090H      ;P1.0 - EXT. INTERRUPT 3/CAPT & COMP 0
INT4      BIT       091H      ;P1.1 - EXT. INTERRUPT 4/CAPT & COMP 1
INT5      BIT       092H      ;P1.2 - EXT. INTERRUPT 5/CAPT & COMP 2
INT6      BIT       093H      ;P1.3 - EXT. INTERRUPT 6/CAPT & COMP 3
INT2      BIT       094H      ;P1.4 - EXT. INTERRUPT 2
T2EX      BIT       095H      ;P1.5 - TIMER 2 EXT. RELOAD TRIGGER INP
CLKOUT    BIT       096H      ;P1.6 - SYSTEM CLOCK OUTPUT
T2        BIT       097H      ;P1.7 - TIMER 2 INPUT
************************************************************************
************************************************************************
for the 83C152/80C152
GRXD      BIT       090H      ;P1.0 - GSC RECEIVER DATA INPUT
GTXD      BIT       091H      ;P1.1 - GSC TRANSMITTER DATA OUTPUT
DEN       BIT       092H      ;P1.2 - DRIVE ENABLE TO ENABLE EXT DRIVE
TXC       BIT       093H      ;P1.3 - GSC EXTERNAL TRANSMIT CLOCK INPU
RXC       BIT       094H      ;P1.4 - GSC EXTERNAL RECEIVER CLOCK INPU
************************************************************************
************************************************************************
CT0I      BIT       090H      ;P1.0 - CAPTURE/TIMER INPUT 0
CT1I      BIT       091H      ;P1.1 - CAPTURE/TIMER INPUT 1
CT2I      BIT       092H      ;P1.2 - CAPTURE/TIMER INPUT 2
CT3I      BIT       093H      ;P1.3 - CAPTURE/TIMER INPUT 3
T2        BIT       094H      ;P1.4 - T2 EVENT INPUT
RT2       BIT       095H      ;P1.5 - T2 TIMER RESET SIGNAL
SCL       BIT       096H      ;P1.6 - SERIAL PORT CLOCK LINE I2C
SDA       BIT       097H      ;P1.7 - SERIAL PORT DATA LINE I2C
************************************************************************
************************************************************************
for the 80C517/80C537
INT3      BIT       090H      ;P1.0 - EXT. INTERRUPT 3/CAPT & COMP 0
INT4      BIT       091H      ;P1.1 - EXT. INTERRUPT 4/CAPT & COMP 1
INT5      BIT       092H      ;P1.2 - EXT. INTERRUPT 5/CAPT & COMP 2
INT6      BIT       093H      ;P1.3 - EXT. INTERRUPT 6/CAPT & COMP 3
INT2      BIT       094H      ;P1.4 - EXT. INTERRUPT 2
T2EX      BIT       095H      ;P1.5 - TIMER 2 EXT. RELOAD TRIGGER INPU
CLKOUT    BIT       096H      ;P1.6 - SYSTEM CLOCK OUTPUT
```

```
T2        BIT       097H      ;P1.7 - TIMER 2 INPUT
*************************************************************************
*************************************************************************
for the 80C452/83C452 and 80C152/83C152
HLD       BIT       095H      ;P1.5 - DMA HOLD REQUEST I/O
HLDA      BIT       096H      ;P1.6 - DMA HOLD ACKNOWLEDGE OUTPUT
*************************************************************************
*************************************************************************
for the 83C751/83C752
INT0      BIT       095H      ;P1.5 - EXTERNAL INTERRUPT 0 INPUT
INT1      BIT       096H      ;P1.6 - EXTERNAL INTERRUPT 1 INPUT
T0        BIT       096H      ;P1.7 - TIMER 0 COUNT INPUT
*************************************************************************

RI        BIT       098H      ;SCON.0 - RECEIVE INTERRUPT FLAG
TI        BIT       099H      ;SCON.1 - TRANSMIT INTERRUPT FLAG
RB8       BIT       09AH      ;SCON.2 - RECEIVE BIT 8
TB8       BIT       09BH      ;SCON.3 - TRANSMIT BIT 8
REN       BIT       09CH      ;SCON.4 - RECEIVE ENABLE
SM2       BIT       09DH      ;SCON.5 - SERIAL MODE CONTROL BIT 2
SM1       BIT       09EH      ;SCON.6 - SERIAL MODE CONTROL BIT 1
SM0       BIT       09FH      ;SCON.7 - SERIAL MODE CONTROL BIT 0

*************************************************************************
for the 83C751/83C752
MASTER    BIT(READ) 099H      ;I2CON.1 - MASTER
STP       BIT(READ) 09AH      ;I2CON.2 - STOP
STR       BIT(READ) 09BH      ;I2CON.3 - START
ARL       BIT(READ) 09CH      ;I2CON.4 - ARBITRATION LOSS
DRDY      BIT(READ) 09DH      ;I2CON.5 - DATA READY
ATN       BIT(READ) 09EH      ;I2CON.6 - ATTENTION
RDAT      BIT(READ) 09FH      ;I2CON.7 - RECEIVE DATA
XSTP      BIT(WRITE)098H      ;I2CON.0 - TRANSMIT STOP
XSTR      BIT(WRITE)099H      ;I2CON.1 - TRANSMIT REPEATED START
CSTP      BIT(WRITE)09AH      ;I2CON.2 - CLEAR STOP
CSTR      BIT(WRITE)09BH      ;I2CON.3 - CLEAR START
CARL      BIT(WRITE)09CH      ;I2CON.4 - CLEAR ARBITRATION LOSS
CDR       BIT(WRITE)09DH      ;I2CON.5 - CLEAR DATA READY
IDLE      BIT(WRITE)09EH      ;I2CON.6 - GO IDLE
CXA       BIT(WRITE)09FH      ;I2CON.7 - CLEAR TRANSMIT ACTIVE
*************************************************************************

EX0       BIT       0A8H      ;IE.0 - EXTERNAL INTERRUPT 0 ENABLE
ET0       BIT       0A9H      ;IE.1 - TIMER 0 INTERRUPT ENABLE
EX1       BIT       0AAH      ;IE.2 - EXTERNAL INTERRUPT 1 ENABLE
ET1       BIT       0ABH      ;IE.3 - TIMER 1 INTERRUPT ENABLE
ES        BIT       0ACH      ;IE.4 - SERIAL PORT INTERRUPT ENABLE

*************************************************************************
for the 83C751/83C752
EI2       BIT       0ACH      ;IE.4 - SERIAL PORT INTERRUPT ENABLE
*************************************************************************
*************************************************************************
for the 8052/8032, 80C154/83C154, 80C252(80C51FA), 80515/80535
ET2       BIT       0ADH      ;TIMER 2 INTERRUPT ENABLE
*************************************************************************
*************************************************************************
for the 80C652/83C652
ES1       BIT       0ADH      ;IE.5 - SERIAL PORT 1 INTERRUPT ENABLE
*************************************************************************
```

```
*************************************************************************
for the 80C252(80C51FA)
EC          BIT         0AEH        ;IE.6 - ENABLE PCA INTERRUPT
*************************************************************************
*************************************************************************
for the 80515/80535
WDT         BIT         0AEH        ;IEN0.6 - WATCHDOG TIMER RESET
*************************************************************************
*************************************************************************
for the 83C552/80C552
ES1         BIT         0ADH        ;IEN0.5 - SERIAL PORT 1 INTERRUPT ENABLE
EAD         BIT         0AEH        ;IEN0.6 - ENABLE A/D INTERRUPT
*************************************************************************
*************************************************************************
for the 80C517/80C537
ET2         BIT         0ADH        ;IEN0.5 - TIMER 2 INTERRUPT ENABLE
WDT         BIT         0AEH        ;IEN0.6 - WATCHDOG TIMER RESET
*************************************************************************

EA          BIT         0AFH        ;IE.7 - GLOBAL INTERRUPT ENABLE
RXD         BIT         0B0H        ;P3.0 - SERIAL PORT RECEIVE INPUT
TXD         BIT         0B1H        ;P3.1 - SERIAL PORT TRANSMIT OUTPUT
INT0        BIT         0B2H        ;P3.2 - EXTERNAL INTERRUPT 0 INPUT
INT1        BIT         0B3H        ;P3.3 - EXTERNAL INTERRUPT 1 INPUT
T0          BIT         0B4H        ;P3.4 - TIMER 0 COUNT INPUT
T1          BIT         0B5H        ;P3.5 - TIMER 1 COUNT INPUT
WR          BIT         0B6H        ;P3.6 - WRITE CONTROL FOR EXT. MEMORY
RD          BIT         0B7H        ;P3.7 - READ CONTROL FOR EXT. MEMORY
PX0         BIT         0B8H        ;IP.0 - EXTERNAL INTERRUPT 0 PRIORITY
PT0         BIT         0B9H        ;IP.1 - TIMER 0 PRIORITY
PX1         BIT         0BAH        ;IP.2 - EXTERNAL INTERRUPT 1 PRIORITY
PT1         BIT         0BBH        ;IP.3 - TIMER 1 PRIORITY
PS          BIT         0BCH        ;IP.4 - SERIAL PORT PRIORITY

*************************************************************************
for the 80C154/83C154
PT2         BIT         0BCH        ;IP.5 - TIMER 2 PRIORITY
PCT         BIT         0BFH        ;IP.7 - INTERRUPT PRIORITY DISABLE
*************************************************************************
*************************************************************************
for the 80C652/
PS1         BIT         0BDH        ;IP.5 - SERIAL PORT 1 PRIORITY
*************************************************************************
*************************************************************************
for the 80C51FA/83C51FA(83C252/80C252)
PT2         BIT         0BDH        ;IP.5 - TIMER 2 PRIORITY
PPC         BIT         0BEH        ;IP.6 - PCA PRIORITY
*************************************************************************
*************************************************************************
for the 80515/80535 and 80C517/80C537
EADC        BIT         0B8H        ;IEN1.0 - A/D CONVERTER INTERRUPT EN
EX2         BIT         0B9H        ;IEN1.1 - EXT. INTERRUPT 2 ENABLE
EX3         BIT         0BAH        ;IEN1.2 - EXT. INT 3/CAPT/COMP INT 0 EN
EX4         BIT         0BBH        ;IEN1.3 - EXT. INT 4/CAPT/COMP INT 1 EN
EX5         BIT         0BCH        ;IEN1.4 - EXT. INT 5/CAPT/COMP INT 2 EN
EX6         BIT         0BDH        ;IEN1.5 - EXT. INT 6/CAPT/COMP INT 3 EN
SWDT        BIT         0BEH        ;IEN1.6 - WATCHDOG TIMER START
EXEN2       BIT         0BFH        ;IEN1.7 - T2 EXT. RELOAD INTER START
IADC        BIT         0C0H        ;IRCON.0 - A/D CONVERTER INTER REQUEST
```

```
IEX2      BIT      0C1H      ;IRCON.1 - EXT. INTERRUPT 2 EDGE FLAG
IEX3      BIT      0C2H      ;IRCON.2 - EXT. INTERRUPT 3 EDGE FLAG
IEX4      BIT      0C3H      ;IRCON.3 - EXT. INTERRUPT 4 EDGE FLAG
IEX5      BIT      0C4H      ;IRCON.4 - EXT. INTERRUPT 5 EDGE FLAG
IEX6      BIT      0C5H      ;IRCON.5 - EXT. INTERRUPT 6 EDGE FLAG
TF2       BIT      0C6H      ;IRCON.6 - TIMER 2 OVERFLOW FLAG
EXF2      BIT      0C7H      ;IRCON.7 - TIMER 2 EXT. RELOAD FLAG
T2IO      BIT      0C8H      ;T2CON.0 - TIMER 2 INPUT SELECT BIT 0
T2I1      BIT      0C9H      ;T2CON.1 - TIMER 2 INPUT SELECT BIT 1
T2CM      BIT      0CAH      ;T2CON.2 - COMPARE MODE
T2R0      BIT      0CBH      ;T2CON.3 - TIMER 2 RELOAD MODE SEL BIT 0
T2R1      BIT      0CCH      ;T2CON.4 - TIMER 2 RELOAD MODE SEL BIT 1
I2FR      BIT      0CDH      ;T2CON.5 - EXT. INT 2 F/R EDGE FLAG
I3FR      BIT      0CEH      ;T2CON.6 - EXT. INT 3 F/R EDGE FLAG
T2PS      BIT      0CFH      ;T2CON.7 - PRESCALER SELECT BIT
***********************************************************************
***********************************************************************
for the 83C552/80C552
PS1       BIT      0BDH      ;IP0.5 - SIO1
PAD       BIT      0BEH      ;IP0.6 - A/D CONVERTER
CMSR0     BIT      0C0H      ;P4.0 - T2 COMPARE AND SET/RESET OUTPUTS
CMSR1     BIT      0C1H      ;P4.1 - T2 COMPARE AND SET/RESET OUTPUTS
CMSR2     BIT      0C2H      ;P4.2 - T2 COMPARE AND SET/RESET OUTPUTS
CMSR3     BIT      0C3H      ;P4.3 - T2 COMPARE AND SET/RESET OUTPUTS
CMSR4     BIT      0C4H      ;P4.4 - T2 COMPARE AND SET/RESET OUTPUTS
CMSR5     BIT      0C5H      ;P4.5 - T2 COMPARE AND SET/RESET OUTPUTS
CMT0      BIT      0C6H      ;P4.6 - T2 COMPARE AND TOGGLE OUTPUTS
CMT1      BIT      0C7H      ;P4.7 - T2 COMPARE AND TOGGLE OUTPUTS
CTI0      BIT      0C8H      ;TM2IR.0 - T2 CAPTURE 0
BIT       0C9H     ;TM2IR.1 - T2 CAPTURE 1
CTI2      BIT      0CAH      ;TM2IR.2 - T2 CAPTURE 2
CTI3      BIT      0CBH      ;TM2IR.3 - T2 CAPTURE 3
CMI0      BIT      0CCH      ;TM2IR.4 - T2 COMPARATOR 0
CMI1      BIT      0CDH      ;TM2IR.5 - T2 COMPARATOR 1
CMI2      BIT      0CEH      ;TM2IR.6 - T2 COMPARATOR 2
T2OV      BIT      0CFH      ;TM2IR.7 - T2 OVERFLOW
***********************************************************************
***********************************************************************
for the RUPI-44
RBP       BIT      0C8H      ;STS.0 - RECEIVE BUFFER PROTECT
AM        BIT      0C9H      ;STS.1 - AUTO/ADDRESSED MODE SELECT
OPB       BIT      0CAH      ;STS.2 - OPTIONAL POLL BIT
BOV          BIT      0CBH      ;STS.3 - RECEIVE BUFFER OVERRUN
SI        BIT      0CCH      ;STS.4 - SIU INTERRUPT FLAG
RTS       BIT      0CDH      ;STS.5 - REQUEST TO SEND
RBE       BIT      0CEH      ;STS.6 - RECEIVE BUFFER EMPTY
TBF       BIT      0CFH      ;STS.7 - TRANSMIT BUFFER FULL
***********************************************************************
***********************************************************************
for the 8052/8032, 80C154/83C154, 80C51FA/83C51FA(83C252/80C252)
CAP2      BIT      0C8H      ;T2CON.0 - CAPTURE OR RELOAD SELECT
CNT2      BIT      0C9H      ;T2CON.1 - TIMER OR COUNTER SELECT
TR2       BIT      0CAH      ;T2CON.2 - TIMER 2 ON/OFF CONTROL
EXEN2     BIT      0CBH      ;T2CON.3 - TIMER 2 EXTERNAL ENABLE FLAG
TCLK      BIT      0CCH      ;T2CON.4 - TRANSMIT CLOCK SELECT
RCLK      BIT      0CDH      ;T2CON.5 - RECEIVE CLOCK SELECT
EXF2      BIT      0CEH      ;T2CON.6 - EXTERNAL TRANSITION FLAG
TF2       BIT      0CFH      ;T2CON.7 - TIMER 2 OVERFLOW FLAG
***********************************************************************
```

```
*************************************************************************
for the 83C152/80C152
EGSRV      BIT       0C8H      ;IEN1.0 - GSC RECEIVE VALID
EGSRE      BIT       0C9H      ;IEN1.1 - GSC RECEIVE ERROR
EDMA0      BIT       0CAH      ;IEN1.2 - DMA CHANNEL REQUEST 0
EGSTV      BIT       0CBH      ;IEN1.3 - GSC TRANSMIT VALID
EDMA1      BIT       0CCH      ;IEN1.4 - DMA CHANNEL REQUEST 1
EGSTE      BIT       0CDH      ;IEN1.5 - GSC TRANSMIT ERROR
*************************************************************************
*************************************************************************
for the 80512/80532
IADC       BIT       0C0H      ;IRCON.0 - A/D CONVERTER INTERRUPT REQ
*************************************************************************

P          BIT       0D0H      ;PSW.0 - ACCUMULATOR PARITY FLAG

*************************************************************************
for the 83C552/80C552
F1         BIT       0D1H      ;PSW.1 - FLAG 1
*************************************************************************
*************************************************************************
for the 80512/80532
F1         BIT       0D1H      ;PSW.1 - FLAG 1
MX0        BIT       0D8H      ;ADCON.0 - ANALOG INPUT CH SELECT BIT 0
MX1        BIT       0D9H      ;ADCON.1 - ANALOG INPUT CH SELECT BIT 1
MX2        BIT       0DAH      ;ADCON.2 - ANALOG INPUT CH SELECT BIT 2
ADM        BIT       0DBH      ;ADCON.3 - A/D CONVERSION MODE
BSY        BIT       0DCH      ;ADCON.4 - BUSY FLAG
BD         BIT       0DFH      ;ADCON.7 - BAUD RATE ENABLE
*************************************************************************

OV         BIT       0D2H      ;PSW.2 - OVERFLOW FLAG
RS0        BIT       0D3H      ;PSW.3 - REGISTER BANK SELECT 0
RS1        BIT       0D4H      ;PSW.4 - REGISTER BANK SELECT 1
F0         BIT       0D5H      ;PSW.5 - FLAG 0
AC         BIT       0D6H      ;PSW.6 - AUXILIARY CARRY FLAG
CY         BIT       0D7H      ;PSW.7 - CARRY FLAG

*************************************************************************
for the 80C51FA/83C51FA(83C252/80C252)
CCF0       BIT       0D8H      ;CCON.0 -PCA MODULE 0 INTERRUPT FLAG
CCF1       BIT       0D9H      ;CCON.1 -PCA MODULE 1 INTERRUPT FLAG
CCF2       BIT       0DAH      ;CCON.2 -PCA MODULE 2 INTERRUPT FLAG
CCF3       BIT       0DBH      ;CCON.3 -PCA MODULE 3 INTERRUPT FLAG
CCF4       BIT       0DCH      ;CCON.4 -PCA MODULE 4 INTERRUPT FLAG
CR         BIT       0DEH      ;CCON.6 - COUNTER RUN
CF         BIT       0DFH      ;PCA COUNTER OVERFLOW FLAG
*************************************************************************
*************************************************************************
for the RUPI-44
SER        BIT       0D8H      ;NSNR.0 - RECEIVE SEQUENCE ERROR
NR0        BIT       0D9H      ;NSNR.1 - RECEIVE SEQUENCE COUNTER-BIT 0
NR1        BIT       0DAH      ;NSNR.2 - RECEIVE SEQUENCE COUNTER-BIT 1
NR2        BIT       0DBH      ;NSNR.3 - RECEIVE SEQUENCE COUNTER-BIT 2
SES        BIT       0DCH      ;NSNR.4 - SEND SEQUENCE ERROR
NS0        BIT       0DDH      ;NSNR.5 - SEND SEQUENCE COUNTER-BIT 0
NS1        BIT       0DEH      ;NSNR.6 - SEND SEQUENCE COUNTER-BIT 1
NS2        BIT       0DFH      ;NSNR.7 - SEND SEQUENCE COUNTER-BIT 2
*************************************************************************
```

```
************************************************************************
for the 80515/80535
MX0       BIT       0D8H      ;ADCON.0 - ANALOG INPUT CH SELECT BIT 0
MX1       BIT       0D9H      ;ADCON.1 - ANALOG INPUT CH SELECT BIT 1
MX2       BIT       0DAH      ;ADCON.2 - ANALOG INPUT CH SELECT BIT 2
ADM       BIT       0DBH      ;ADCON.3 - A/D CONVERSION MODE
BSY       BIT       0DCH      ;ADCON.4 - BUSY FLAG
CLK       BIT       0DEH      ;ADCON.5 - SYSTEM CLOCK ENABLE
BD        BIT       0DFH      ;ADCON.7 - BAUD RATE ENABLE
************************************************************************
************************************************************************
for the 80C652/83C652
CR0       BIT       0D8H      ;S1CON.0 - CLOCK RATE 0
CR1       BIT       0D9H      ;S1CON.1 - CLOCK RATE 1
AA        BIT       0DAH      ;S1CON.2 - ASSERT ACKNOWLEDGE
SI        BIT       0DBH      ;S1CON.3 - SIO1 INTERRUPT BIT
STO       BIT       0DCH      ;S1CON.4 - STOP FLAG
STA       BIT       0DDH      ;S1CON.5 - START FLAG
ENS1      BIT       0DEH      ;S1CON.6 - ENABLE SIO1
************************************************************************
************************************************************************
for the 83C152/80C152
DMA       BIT       0D8H      ;TSTAT.0 - DMA SELECT
TEN       BIT       0D9H      ;TSTAT.1 - TRANSMIT ENABLE
TFNF      BIT       0DAH      ;TSTAT.2 - TRANSMIT FIFO NOT FULL
TDN       BIT       0DBH      ;TSTAT.3 - TRANSMIT DONE
TCDT      BIT       0DCH      ;TSTAT.4 - TRANSMIT COLLISION DETECT
UR        BIT       0DDH      ;TSTAT.5 - UNDERRUN
NOACK     BIT       0DEH      ;TSTAT.6 - NO ACKNOWLEDGE
LNI       BIT       0DFH      ;TSTAT.7 - LINE IDLE
HBAEN     BIT       0E8H      ;RSTAT.0 - HARDWARE BASED ACKNOWLEDGE EN
GREN      BIT       0E9H      ;RSTAT.1 - RECEIVER ENABLE
RFNE      BIT       0EAH      ;RSTAT.2 - RECEIVER FIFO NOT EMPTY
RDN       BIT       0EBH      ;RSTAT.3 - RECEIVER DONE
CRCE      BIT       0ECH      ;RSTAT.4 - CRC ERROR
AE        BIT       0EDH      ;RSTAT.5 - ALIGNMENT ERROR
RCABT     BIT       0EEH      ;RSTAT.6 - RCVR COLLISION/ABORT DETECT
OR        BIT       0EFH      ;RSTAT.7 - OVERRUN
PGSRV     BIT       0F8H      ;IPN1.0 - GSC RECEIVE VALID
PGSRE     BIT       0F9H      ;IPN1.1 - GSC RECEIVE ERROR
PDMA0     BIT       0FAH      ;IPN1.2 - DMA CHANNEL REQUEST 0
PGSTV     BIT       0FBH      ;IPN1.3 - GSC TRANSMIT VALID
PDMA1     BIT       0FCH      ;IPN1.4 - DMA CHANNEL REQUEST 1
PGSTE     BIT       0FDH      ;IPN1.5 - GSC TRANSMIT ERROR
************************************************************************
************************************************************************
for the 80C452/83C452
OFRS      BIT       0E8H      ;SLCON.0 - OUTPUT FIFO CH REQ SERVICE
IFRS      BIT       0E9H      ;SLCON.1 - INPUT FIFO CH REQ SERVICE
FRZ       BIT       0EBH      ;SLCON.3 - ENABLE FIFO DMA FREEZE MODE
ICOI      BIT       0ECH      ;SLCON.4 - GEN INT WHEN IMMEDIATE COMMAN
                              ;OUT REGISTER IS AVAILABLE
ICII      BIT       0EDH      ;SLCON.5 - GEN INT WHEN A COMMAND IS
                              ;WRITTEN TO IMMEDIATE COMMAND IN REG
OFI       BIT       0EEH      ;SLCON.6 - ENABLE OUTPUT FIFO INTERRUPT
IFI       BIT       0EFH      ;SLCON.7 - ENABLE INPUT FIFO INTERRUPT
EFIFO     BIT       0F8H      ;IEP.0 - FIFO SLAVE BUS I/F INT EN
PDMA1     BIT       0F9H      ;IEP.1 - DMA CHANNEL REQUEST 1
PDMA0     BIT       0FAH      ;IEP.2 - DMA CHANNEL REQUEST 0
EDMA1     BIT       0FBH      ;IEP.3 - DMA CHANNEL 1 INTERRUPT ENABLE
```

```
EDMA0     BIT       0FCH      ;IEP.4 - DMA CHANNEL 0 INTERRUPT ENABLE
PFIFO     BIT       0FDH      ;IEP.5 - FIFO SLAVE BUS I/F INT PRIORITY
***********************************************************************
***********************************************************************
for the 80C451/83C451
IBF       BIT       0E8H      ;CSR.0 - INPUT BUFFER FULL
OBF       BIT       0E9H      ;CSR.1 - OUTPUT BUFFER FULL
IDSM      BIT       0EAH      ;CSR.2 - INPUT DATA STROBE
OBFC      BIT       0EBH      ;CSR.3 - OUTPUT BUFFER FLAG CLEAR
MA0       BIT       0ECH      ;CSR.4 - AFLAG MODE SELECT
MA1       BIT       0EDH      ;CSR.5 - AFLAG MODE SELECT
MB0       BIT       0EEH      ;CSR.6 - BFLAG MODE SELECT
MB1       BIT       0EFH      ;CSR.7 - BFLAG MODE SELECT
***********************************************************************
***********************************************************************
for the 83C751/83C752
CTO       BIT(READ) 0D8H      ;I2CFG.0 - CLOCK TIMING 0
CT1       BIT(READ) 0D9H      ;I2CFG.1 - CLOCK TIMING 1
T1RUN     BIT(READ) 0DCH      ;I2CFG.4 - START/STOP TIMER 1
MASTRQ    BIT(READ) 0DEH      ;I2CFG.6 - MASTER I2C
SLAVEN    BIT(READ) 0DFH      ;I2CFG.7 - SLAVE I2C
CT0       BIT(WRITE)0D8H      ;I2CFG.0 - CLOCK TIMING 0
CT1       BIT(WRITE)0D9H      ;I2CFG.1 - CLOCK TIMING 1
TIRUN     BIT(WRITE)0DCH      ;I2CFG.4 - START/STOP TIMER 1
CLRTI     BIT(WRITE)0DDH      ;I2CFG.5 - CLEAR TIMER 1 INTERRUPT FLAG
MASTRQ    BIT(WRITE)0DEH      ;I2CFG.6 - MASTER I2C
SLAVEN    BIT(WRITE)0DFH      ;I2CFG.7 - SLAVE I2C
RSTP      BIT(READ) 0F8H      ;I2STA.0 - XMIT STOP CONDITION
RSTR      BIT(READ) 0F9H      ;I2STA.1 - XMIT REPEAT STOP COND.
MAKSTP    BIT(READ) 0FAH      ;I2STA.2 - STOP CONDITION
MAKSTR    BIT(READ) 0FBH      ;I2STA.3 - START CONDITION
XACTV     BIT(READ) 0FCH      ;I2STA.4 - XMIT ACTIVE
XDATA     BIT(READ) 0FDH      ;I2STA.5 - CONTENT OF XMIT BUFFER
RIDLE     BIT(READ) 0FEH      ;I2STA.6 - SLAVE IDLE FLAG
***********************************************************************
***********************************************************************
for the 83C552/80C552
CR0       BIT       0D8H      ;S1CON.0 - CLOCK RATE 0
CR1       BIT       0D9H      ;S1CON.1 - CLOCK RATE 1
AA        BIT       0DAH      ;S1CON.2 - ASSERT ACKNOWLEDGE
SI        BIT       0DBH      ;S1CON.3 - SERIAL I/O INTERRUPT
STO       BIT       0DCH      ;S1CON.4 - STOP FLAG
STA       BIT       0DDH      ;S1CON.5 - START FLAG
ENS1      BIT       0DEH      ;S1CON.6 - ENABLE SERIAL I/O
ECT0      BIT       0E8H      ;IEN1.0 - ENABLE T2 CAPTURE 0
ECT1      BIT       0E9H      ;IEN1.1 - ENABLE T2 CAPTURE 1
ECT2      BIT       0EAH      ;IEN1.2 - ENABLE T2 CAPTURE 2
ECT3      BIT       0EBH      ;IEN1.3 - ENABLE T2 CAPTURE 3
ECM0      BIT       0ECH      ;IEN1.4 - ENABLE T2 COMPARATOR 0
ECM1      BIT       0EDH      ;IEN1.5 - ENABLE T2 COMPARATOR 1
ECM2      BIT       0EEH      ;IEN1.6 - ENABLE T2 COMPARATOR 2
ET2       BIT       0EFH      ;IEN1.7 - ENABLE T2 OVERFLOW
PCT0      BIT       0F8H      ;IP1.0 - T2 CAPTURE REGISTER 0
PCT1      BIT       0F9H      ;IP1.1 - T2 CAPTURE REGISTER 1
PCT2      BIT       0FAH      ;IP1.2 - T2 CAPTURE REGISTER 2
PCT3      BIT       0FBH      ;IP1.3 - T2 CAPTURE REGISTER 3
PCM0      BIT       0FCH      ;IP1.4 - T2 COMPARATOR 0
PCM1      BIT       0FDH      ;IP1.5 - T2 COMPARATOR 1
PCM2      BIT       0FEH      ;IP1.6 - T2 COMPARATOR 2
PT2       BIT       0FFH      ;IP1.7 - T2 OVERFLOW
```

```
************************************************************************
************************************************************************
for the 80C517/80C537
F1          BIT         0D1H        ;PSW.1 - FLAG 1
MX0         BIT         0D8H        ;ADCON0.0 - ANALOG INPUT CH SELECT BIT 0
MX1         BIT         0D9H        ;ADCON0.1 - ANALOG INPUT CH SELECT BIT 1
MX2         BIT         0DAH        ;ADCON0.2 - ANALOG INPUT CH SELECT BIT 2
ADM         BIT         0DBH        ;ADCON0.3 - A/D CONVERSION MODE
BSY         BIT         0DCH        ;ADCON0.4 - BUSY FLAG
CLK         BIT         0DEH        ;ADCON0.5 - SYSTEM CLOCK ENABLE
BD          BIT         0DFH        ;ADCON0.7 - BAUD RATE ENABLE
************************************************************************
************************************************************************
for the 80C154/83C154
ALF         BIT         0F8H        ;IOCON.0 - CPU POWER DOWN MODE CONTROL
P1F         BIT         0F9H        ;IOCON.1 - PORT 1 HIGH IMPEDANCE
P2F         BIT         0FAH        ;IOCON.2 - PORT 2 HIGH IMPEDANCE
P3F         BIT         0FBH        ;IOCON.3 - PORT 3 HIGH IMPEDANCE
IZC         BIT         0FCH        ;IOCON.4 - 10K TO 100 K OHM SWITCH (P1-3)
SERR        BIT         0FDH        ;IOCON.5 - SERIAL PORT RCV ERROR FLAG
T32         BIT         0FEH        ;IOCON.6 - 32 BIT TIMER SWITCH
WDT         BIT         0FFH        ;IOCON.7 - WATCHDOG TIMER CONTROL
************************************************************************
```

APPENDIX C - RESERVED SYMBOLS

The following is a list of reserved symbols used by the Cross Assembler. These symbols cannot be redefined.

A	AB	ACALL	ADD
ADDC	AJMP	AND	ANL
AR0	AR1	AR2	AR3
AR4	AR5	AR6	AR7
BIT	BSEG	C	CALL
CJNE	CLR	CODE	CPL
CSEG	DA	DATA	DB
DBIT	DEC	DIV	DJNZ
DPTR	DS	DSEG	DW
END	EQ	EQU	GE
GT	HIGH	IDATA	INC
ISEG	JB	JBC	JC
JMP	JNB	JNC	JNZ
JZ	LCALL	LE	LJMP
LOW	LT	MOD	MOV
MOVC	MOVX	MUL	NE
NOP	NOT	OR	ORG
ORL	PC	POP	PUSH
R0	R1	R2	R3
R4	R5	R6	R7
RET	RETI	RL	RLC
RR	RRC	SET	SETB
SHL	SHR	SJMP	SUBB
SWAP	USING	XCH	XCHD
XDATA	XOR	XRL	XSEG

APPENDIX D - CROSS ASSEMBLER CHARACTER SET

CHARACTER NAME	PRINTABLE FORM	ASCII CODE	
		HEX	DECIMAL
Horizontal Tab		09	9
Line Feed		0A	10
Carriage Return		0D	13
Space		20	32
Exclamation Point	!	21	33
Pound Sign	#	23	35
Dollar Sign	$	24	36
Percent Sign	%	25	37
Ampersand	&	26	38
Apostrophe	'	27	39
Left Parenthesis	(	28	40
Right Parenthesis	)	29	41
Asterisk	*	2A	42
Plus sign	+	2B	43
Comma	,	2C	44
Hyphen	-	2D	45
Period	.	2E	46
Slash	/	2F	47
Number 0	0	30	48
" 1	1	31	49
" 2	2	32	50
" 3	3	33	51
" 4	4	34	52
" 5	5	35	53
" 6	6	36	54
" 7	7	37	55
" 8	8	38	56
" 9	9	39	57
Colon	:	3A	58
Semi-colon	;	3B	59
Left Angle Bracket	<	3C	60
Equal Sign	=	3D	61
Right Angle Bracket	>	3E	62
Question Mark	?	3F	63
At Sign	@	40	64
Upper Case A	A	41	65
" " B	B	42	66
" " C	C	43	67
" " D	D	44	68
" " E	E	45	69
" " F	F	46	70
" " G	G	47	71
" " H	H	48	72

CHARACTER NAME	PRINTABLE FORM	ASCII CODE	
		HEX	DECIMAL
Upper Case I	I	49	73
" " J	J	4A	74
" " K	K	4B	75
" " L	L	4C	76
" " M	M	4D	77
" " N	N	4E	78
" " O	O	4F	79
" " P	P	50	80
" " Q	Q	51	81
" " R	R	52	82
" " S	S	53	83
" " T	T	54	84
" " U	U	55	85
" " V	V	56	86
" " W	W	57	87
" " X	X	58	88
" " Y	Y	59	89
" " Z	Z	5A	90
Underscore	_	5F	95
Lower Case A	a	61	97
" " B	b	62	98
" " C	c	63	99
" " D	d	64	100
" " E	e	65	101
" " F	f	66	102
" " G	g	67	103
" " H	h	68	104
" " I	I	69	105
" " J	j	6A	106
" " K	k	6B	107
" " L	l	6C	108
" " M	m	6D	109
" " N	n	6E	110
" " O	o	6F	111
" " P	p	70	112
" " Q	q	71	113
" " R	r	72	114
" " S	s	73	115
" " T	t	74	116
" " U	u	75	117
" " V	v	76	118
" " W	w	77	119
" " X	x	78	120
" " Y	y	79	121
" " Z	z	7A	122

INDEX